ACTIVE PERCEPTION AND PLANNING FOR MODULAR SELF-RECONFIGURABLE ROBOTS

A Dissertation

Presented to the Faculty of the Graduate School

of Cornell University

in Partial Fulfillment of the Requirements for the Degree of

Doctor of Philosophy

by

Jonathan Daudelin

August 2018

ACTIVE PERCEPTION AND PLANNING FOR MODULAR SELF-RECONFIGURABLE ROBOTS

Jonathan Daudelin, Ph.D.

Cornell University 2018

Modular robots have the unique ability to reconfigure their shape and capabilities to adapt to various challenges in the environment. In order to perform tasks autonomously in unknown environments, active perception and planning algorithms are required that can leverage their adaptive capabilities. This work presents several such perception and planning tools. An novel, probabilistic object reconstruction algorithm is presented that allows a generic mobile robot (such as a modular robot) intelligently position a 3D sensor to explore unknown objects in its environment. Then, it presents fully autonomous, perception-informed systems for modular self-reconfigurable robots (MSRRs) that enable them to explore, dynamically adapt to their environment, and even augment their environment to perform high-level tasks. Finally, it presents an end-to-end path planning framework for MSRR systems that enables them to reconfigure between multiple morphologies and use multiple gaits in order to traverse and plan optimal paths over challenging terrain.

BIOGRAPHICAL SKETCH

Jonathan Daudelin received his B.S. in Mechanical Engineering from New Jersey Institute of Technology in 2013, graduating with highest honors. He received his M.S. degree from Cornell University in Mechanical Engineering in 2016. His research focuses on active sensing and planning for information gain in autonomous robots. He received the National Science Foundation Graduate Research Fellowship, and is a member of Tau Beta Pi.

This work is dedicated to the most beautiful girl in the world, my wife Larissa. You were willing to give up everything, including your country and Ph.D., to start a life with me. For your love, the rest of my life and all of my love is yours.

ACKNOWLEDGEMENTS

Any credit I might receive for this work belongs to God, who brought me to Cornell, enabled me to complete my Ph.D., and has blessed my life in all areas so much more than I could ask for or deserve.

I would like to thank Professor Mark Campbell for being an amazing advisor. My experience in Cornell's Ph.D. program has been so enjoyable, educational, impactful, and even humorous due to his mentorship, encouragement, leadership, and instruction. I would also like to thank Professor Hadas Kress-Gazit for being a great PI and committee member, and for "adopting" me into her group as well.

I would like to thank my fellow graduate students in the Campbell and Kress-Gazit research groups for their comradery, support, and occasional sarcastic jokes. In particular I would like to thank Ganyuan (Jim) Jing and Tarik Tosun (from the University of Pennsylvania), who spent long hours and late nights working with me on our modular robotics projects. I couldn't have asked for a better team! I would also like to thank Alex Ivanov for constructive arguments and research advice, combined with constant insults and banter.

I want to thank Douglas and Vickey Daudelin, the best Dad and Mom in the world. They constantly sacrificed their interests to do what was best for me and my siblings, and prepared me so well for higher education and life in general. I want to thank my siblings: David, Isaac, Timothy, Elizabeth, Daniel, John, Anna, Cassia, and Peter. Best pack of siblings around! I wouldn't have traded growing up with you all for anything in the world. I would also like to thank my grandparents, Abe and Sylvia Daudelin. They always worked hard and put family first, and much of the credit for this work is due to the foundations they laid many years ago.

Finally, I want to thank my wonderful wife, Larissa. Thank you for your perfect love, companionship, and support; you are my motivation and the love of my life.

TABLE OF CONTENTS

[1]This work was conducted in collaboration with Gangyuan Jing from Cornell University and Tarik Tosun from University of Pennsylvania

[2]This work was conducted in collaboration with Gangyuan Jing from Cornell University and Tarik Tosun from University of Pennsylvania

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

Autonomous mobile robots perform many tasks, such as grasping and inspection, that may require complete models of 3D objects in the environment. If little or no knowledge about an object is known *a priori*, the robot must take sensor measurements from strategically determined viewpoints in order to reconstruct a 3D model of the object. This can be especially useful in modular self-reconfigurable robots (MSRRs), which have the potential to adapt to challenging terrain via reconfiguration in order to reach target destinations. However, in order to plan paths over complicated objects in the environment, a map of the environment - especially such objects - must be obtained via exploration, and a planning framework must be implemented that characterizes the terrain in terms of the robot's navigation abilities.

Chapter 2 begins by addressing the exploration problem. It proposes an autonomous object reconstruction approach for mobile robots that is very general, with no assumptions about object shape or size, such as a bounding box or predetermined set of candidate viewpoints. A probabilistic, volumetric method for determining the optimal next best view is developed based on a partial model of a 3D object of unknown shape and size. The proposed method integrates an object probability characteristic to determine sensor views that incrementally reconstruct a 3D model of the object. Experiments in simulation and on a real world robot validate the work and compare it to the state of the art.

Chapters 3 and 4 present the first modular robot system capable of autonomously completing high-level tasks by reactively reconfiguring to meet the needs of a perceived, *a priori* unknown environment. Chapter 3 presents a system

that tightly integrates perception, high-level planning, and modular hardware, and is validated in three hardware demonstrations. Based on a high-level task specification, the modular robot autonomously explores an unknown environment (using the object reconstruction algorithm from Chapter 2), decides when and how to reconfigure, and manipulates objects to complete its task. Three real-world experiments using different high-level tasks and environmental challenges demonstrate the effectiveness and adaptive capabilities of the system. These results mark a milestone in the field and represents the state of the art of autonomy in self-reconfigurable modular robotics. Chapter 4 extends the work in Chapter 3 to enable modular robots to augment their environments with building blocks in order to extend their navigation abilities. Two hardware experiments demonstrate the validity of the system.

Chapter 5 presents a complete optimal path planning framework for generic modular self reconfigurable robots (MSRRs) that enables navigation over complex and difficult terrain by leveraging MSRRs' ability to change their morphology and action capabilities. Such terrain includes features such as ledges, undulating surfaces, bridges and narrow corridors. Using an elevation map of the environment as input, the presented approach uses a sliding window convolutional neural network (CNN) classifier to characterize the terrain into a 2D grid of traversable features and obstacles. A 4D lattice planner then uses the characterized terrain map to plan paths to desired goals using action primitives for each traversable feature. The planner finds optimal paths to the goal in realtime, accounting for different locomotion costs for various robot morphologies and actions as well as the cost of reconfiguring to a different morphology. Hybrid experiments demonstrate the ability of the planner with a simulated SMORES modular robot system, using sensor measurements from a real-world environment.

CHAPTER 2

AN ADAPTABLE, PROBABILISTIC, NEXT BEST VIEW ALGORITHM FOR RECONSTRUCTION OF UNKNOWN 3D OBJECTS

2.1 Introduction

An important aspect of autonomous mobile robots is their ability to interact with the environment. In order to achieve this, robots must sense and perceive their environment, including sometimes the shapes of objects. For example, a robot may need to know the shape of an object for classification, or may need to inspect a large object to find an item of interest. Many times, objects are not known *a priori*, requiring the robot to first explore the object via sensor measurements from multiple viewpoints. During exploration of an unknown environment, objects of interest may also have unknown size/extent. We propose an approach for autonomous exploration and reconstruction of objects with no *a priori* assumptions on their shape or bounding box. We also propose a fully probabilistic method for determining the Next Best View (NBV) for sensor placement, which forms the basis for the autonomous reconstruction algorithm. The method uses an entropy-based cost function in conjunction with a 3D occupancy grid to predict the expected information gain for measurements from candidate viewpoints based on the current object representation. The NBV algorithm weighs entropy calculations for each voxel by its probabilitiy of belonging to the object and being visible from a candidate viewpoint. This allows the method to be used with no assumption on object size.

Any partial view of an object is required to begin reconstruction. The algorithm

iteratively selects a NBV based on the current model of the object and attempts to navigate to that viewpoint while continuously collecting new sensor measurements to improve the object reconstruction. This process is repeated until object reconstruction is complete. The NBV search space is dynamically adjusted as the object is reconstructed and is permitted to include unknown space. An information-based termination criterion is used to let the algorithm determine when the object is fully reconstructed.

The paper is outlined as follows. Section II discusses related work. Section III presents the information-based NBV cost function. Section IV presents the reconstruction algorithm. Section V presents results and comparison to other work in simulation experiments, and Section VI presents results from an experiment with a robot in the real world.

2.2 Related Work

Next Best View planning intelligently determines new viewpoints for taking sensor measurements that maximize information collection from the environment. Common examples of desired information are a 3D reconstruction of an object or a map of the environment. Three types of applications that make use of NBV algorithms are: detailed 3D model reconstruction, environment exploration and mapping, and inspection tasks where basic geometry of the object is known *a priori*.

In 3D object reconstruction applications, it is commonly assumed that the object is enclosed in a volume that limits the position and extent of the object. An enclosing sphere has been used, with candidate viewpoints sampled from various positions on the sphere [25] [18] [58] [1] [62]. Other methods assume a bounding

box around a workspace such as a turntable. Pito et al. samples points from a cylinder surrounding a turntable from which to take measurements with a laser stripe scanner [44]. These and similar methods are used for generating detailed 3D reconstructions of small objects that can be grasped by robotic arms or placed on tables inside the workspace of a scanner or sensor mounted on a robotic arm. However, not only does this enclosing volume assumption enforce a limitation on the size of the object, it also assumes that the whole object is visible from viewpoints on the enclosing sphere or cylinder, which is not necessarily true for non-convex objects. Kriegel et al. does not limit candidate viewpoints to a sphere, but still assumes a bounding box containing the object and assumes all other areas in the robot workspace (robotic arm) are free [27].

Dunn et al. and Rainville et al. present NBV strategies for autonomous mobile robots to reconstruct 3D objects without assuming enclosing spheres [11] [8]. However, these methods rely on *a priori* models of the objects for calculating information gain from candidate viewpoints. This work presents a novel approach that makes no *a priori* assumptions about the object shape except the requirement of an initial partial view of the object to start the algorithm.

NBV algorithms designed for mobile robot exploration and mapping generally make no assumption about the size or shape of objects in the environment. However, they also generally make no distinction between different types of unknown volumes in the environment; i.e. the object and the ground. Instead, these algorithms seek to maximize the volume of unknown space visible from the next viewpoint. For example, [61] and [57] search for the closest frontier between known and unknown space and select that as the NBV. Thus, these methods are not well-suited to obtaining representations of specific 3D objects in the environment.

Krainin et al. predicts the information gain (decrease in entropy) for visible portions of the object that have been determined occupied from previous measurements, and assigns a high information gain for visible portions of the object that form boundaries between occupied surfaces and unknown areas [25]. The algorithm then finds the candidate viewpoint that maximizes this predicted information gain. But these information gains from unknown volumes are arbitrarily assigned, not based in probability theory. In addition, the contribution of other unknown volumes to the total information gain is not considered. Potthast et al. uses an entropy-based cost function in combination with a 3D occupancy grid, but uses a bounding volume of a tabletop for containing the object and robot workspace [45]. Vasquez-Gomez et al. [21] uses an octree volumetric method for reconstructing an object and keeps track of all unknown, free, and occupied voxels for calculating the NBV. However, the cost is a function of the number of unknown voxels expected to be visible from candidate viewpoints, and a probabilistic method is not used to predict information gain from new viewpoints. Vasquez-Gomez et al. [56] and recently Isler et al. [22] use volumetric, information gain-based NBV methods in active reconstruction systems for mobile robots, but still assume an enclosing bounding box with a predetermined set of candidate viewpoints, such as a table workspace or enclosing cylinder.

2.2.1 Novel Contributions

This work presents the Adaptable, Probabilistic Object Reconstruction Algorithm (APORA). To the authors' knowledge, APORA is the first complete autonomous 3D object reconstruction algorithm that adapts to objects of any size. It uses a novel information-based view quality function that uses a fully probabilistic

method to predict information gain from candidate viewpoints. In a comparison study, the algorithm performed similar to or better than the recent algorithm by Isler et al. ([22]), and does not require any assumptions on object size as some existing formulations require. (E.g. [45]). A highly efficient, fast implementation of the cost function calculation is also presented that enables the reconstruction algorithm to search candidate viewpoint sets that are several orders of magnitude larger in size than state of the art methods. This enables the proposed algorithm to reconstruct complex and large objects that require much larger candidate viewpoint sets to achieve complete coverage.

To adapt to objects of different sizes, the proposed work presents a novel method for dynamically generating searchspaces of candidate viewpoints based on current object information. These viewpoints are not restricted to known free space in the environment. To account for an unknown environment and viewpoints in unknown space, the proposed work includes a navigation component for dynamic path planning and obstacle avoidance. This component enables the algorithm to determine if a NBV becomes unreachable and accordingly compute a new NBV based on the updated map. The inclusion of this component is novel from existing contributions, which assume bounds on the object of interest and a known clear workspace surrounding it for the configuration space of the robot, (e.g. [27] [28] [56] [21] [22]).

2.3 View Quality Function

Before describing the general architecture of APORA, we present the information-based view quality function. This function takes into account all portions of the

environment in calculating information gain, including empty, occupied, and unknown spaces. It also considers expected self-occlusion from candidate viewpoints, not only from parts of the object already measured, but also from unknown portions of the environment.

2.3.1 Information Gain

As mentioned, the view quality function estimates the information gain that could be obtained from candidate viewpoints. To help accomplish this, a volumetric occupancy grid called Octomap is used to store a probabilistic representation of the object and environment [19]. Octomap is a memory-efficient 3D occupancy grid framework that uses an octree data structure to divide 3D space into hierarchical nodes representing voxels. Occupied, empty, and unknown areas in the environment are tracked. Octomap stores probabilistic occupancy information for each voxel based on integrated measurements from a sensor model.

The quality of a viewpoint is defined as the expected total information gain that would be obtained by taking a new sensor measurement from that viewpoint. The total information gain of an object representation can be found by summing the changes in entropy in each voxel of the object:

$$\sum_{i \in \mathcal{O}} H(m_i|z^{1:k}) - H(m_i|z^{1:k+1}) \tag{2.1}$$

where z^{k+1} is a new measurement, i is a cell index, $\mathcal{O}$ is the set of all voxels occupied by the object of interest, and $H(m_i^k)$ is the entropy of the random binary occupancy variable m_i corresponding to voxel i given measurements $z^{1:k}$. A higher

value of information gain corresponds to a higher certainty about the occupancy of voxels representing the object. Entropy in a mutual information context has been used for calculating expected information gain in robotics applications such as [3].

In general, when considering the NBV for a partially known object in a partially known space, the true set of voxels that correspond to the object is not known with certainty. In addition, the entropy of each voxel after taking a future measurement is clearly uncertain. To account for these uncertanties, the total information gain from new measurement z^{k+1} is defined as follows:

$$\mathcal{I}(z^{k+1}) \triangleq \sum_{i \in \mathcal{M}} [H(m_i|z^{1:k}) - H(m_i|z^{1:k+1})] v_i \cdot o_i \tag{2.2}$$

where v_i is a random binary variable that indicates whether voxel i receives a measurement from z^{k+1} ($v_i = 1$ if a measurement is received, otherwise $v_i = 0$), and o_i indicates whether voxel i belongs to $\mathcal{O}$ ($o_i = 1$ if cell $i \in \mathcal{O}$, otherwise $o_i = 0$). Note that the summation is taken over all voxels in the map $\mathcal{M}$, with o_i and v_i being used as indicator variables to select the appropriate voxels to include in the calculation.

The variables v_i, o_i are random because of the inherent uncertainty in whether each voxel belongs to $\mathcal{O}$, and whether each voxel would receive a measurement from the candidate viewpoint $\mathcal{X}$.

To account for the uncertainty in v_i, o_i, the viewpoint quality $Q(\mathcal{X}, \mathcal{M})$ of a candidate viewpoint $\mathcal{X}$ given a volumetric map $\mathcal{M}$ is defined as the expectation of information gain:

$$Q(\mathcal{X}, \mathcal{M}) \triangleq E[\mathcal{I}(z^{k+1})] =$$

$$\sum_i [H(m_i|z^{1:k}) - H(m_i|z^{1:k+1})] \cdot P(v_i = 1) \cdot P(o_i = 1) \tag{2.3}$$

where z^{k+1} is taken from candidate viewpoint $\mathcal{X}$. Since the summand is 0 when either v_i or o_i are 0, this expectation is equivalent to replacing the indicator variables by the probabilities that each variable is true. In the rest of the paper, $P(v_i = 1)$ and $P(o_i = 1)$ are denoted as $P(v_i)$ and $P(o_i)$ for brevity. The value $Q(\mathcal{X}, \mathcal{M})$ is the predicted information gain of the cost function at viewpoint $\mathcal{X}$. The next section formulates the probabilities $P(v_i)$ and $P(o_i)$.

2.3.2 Occlusion and Object Probabilities

The true probability that cell i belongs to the set of cells representing the object $\mathcal{O}$, $P(o_i)$, involves interdependencies between all cells, making solutions computationally intractable. To enable computational tractability, two assumptions are made about the distribution. For cells that have received sensor measurements, it is assumed that the probability of belonging to the object is equivalent to the probability of occupancy, or $P(o_i) = P(m_i)$. For unknown cells that have received no measurements, it is assumed that $P(o_i)$ depends only on other cells that have received measurements and is independent of other unknown cells. It is assumed that this probability decays with distance from *frontier cells*. Many functions can provide this attribute, but for convenience a squared exponential decay function was chosen:

$$P(o_i) \approx \cdot e^{-(\alpha \cdot d_i^F)^2} \tag{2.4}$$

where d_i^F is the euclidean distance between cell i and the nearest *frontier cell*, and α is a variable set by the user (by default $\alpha = 2$). Frontier cells are unknown cells that border both empty and occupied cells, and are therefore at the edges or "frontiers" of the currently known representation of the object.

Figure 1 shows a 2D example of a partially measured object and the resulting frontier cells. Black cells are occupied, light blue are free, and dark blue are unknown. The red shading on the unknown cells close to the occupied cells indicates their estimated probability of belonging to the object, $P(o_i)$. From Equation 4, the frontier cells have the highest probability (among unknown cells) of belonging to the object. This follows from the assumption that a 3D object is continuous and closed in the discretized occupancy grid representation, and thus there cannot be any frontier cells in a complete representation of the object. As the distance between an unknown cell and the closest frontier cell increases, the probability that the cell belongs to the object decreases exponentially. The factor α can be adjusted to make the algorithm more suited to making predictions for larger or smaller objects, since α determines how fast $P(o_i)$ decays with distance from frontier cells.

The probability that a cell receives a measurement from the candidate viewpoint, $P(v_i)$, takes into account self-occlusion by the object and the range of the sensor. It is assumed that a cell receives a measurement if the cell is in the sensor's range and there is a clear line of sight from the sensor to the cell. In the discretized uncertain occupancy grid representation of the object, this probability is given by the following:

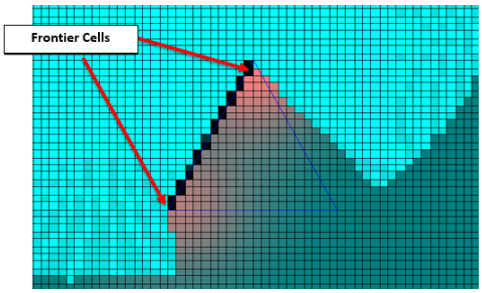


Figure 2.1: An example of a partial representation of a triangle object, showing the frontier cells, and $P(o_i)$ for unknown cells (red shading).

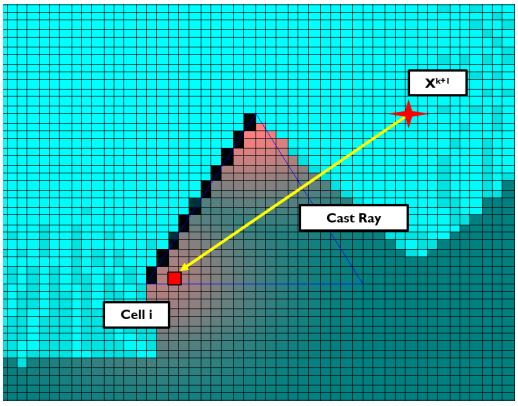


Figure 2.2: Virtual ray cast from candidate viewpoint X^{k+1} and cell i. All cells intersected by the ray are used to calculate $P(v_i)$.

$$P(v_i) = \prod_{i \in \mathcal{R}} (1 - P(o_i)) \tag{2.5}$$

where $\mathcal{R}$ is the set of cells traversed by the ray from the candidate viewpoint $\mathcal{X}$ to cell i. Thus, $P(v_i)$ is the joint probability that all cells between the candidate viewpoint and cell i along the sensor's line of sight are not occupied (no occlusions). Figure 2 illustrates the virtual ray between a candidate viewpoint and a cell. All cells intersected by the yellow ray compose the set $\mathcal{R}$.

With the ability to rank the quality of candidate viewpoints, a NBV search algorithm can be made. The algorithm iteratively chooses a NBV for the sensor based on the current representation of the map $\mathcal{M}$ and a search space $\mathcal{S}$ of

candidate viewpoints as follows:

$$\mathcal{X}^{k+1}(\mathcal{S}, \mathcal{M}^k) = \arg\max_{\mathcal{X} \in \mathcal{S}} Q(\mathcal{X}, \mathcal{M}^k) \tag{2.6}$$

2.3.3 Ground Plane Rejection

In order to keep from reconstructing the entire ground plane along with the object of interest, it is necessary to have some method of distinguishing and rejecting voxels belonging to the ground plane to prevent them from contributing to information gain calculations. This is a problem that must be addressed by any object reconstruction algorithm. This can be done with any ground plane recognition algorithm. The experiments conducted in this work use a simple method that ignores voxels below a threshold height in the volumetric map when calculating view quality.

2.3.4 CPU Time Optimization

In order to make the algorithm fast enough to perform online searching, the implementation of calculating the cost function can be optimized to be computationally efficient while maintaining the theoretical calculations. Values for ΔH and $P(o)$ are calculated once for the unknown voxels near frontier voxels and stored in a secondary occupancy grid to avoid repeat calculations for different candidate viewpionts. Furthermore, these calculations are only performed for unknown voxels close enough to the current object representation to have non-negligible values for $P(o)$. Instead of repeating thousands of ray tracing operations for each can-

didate viewpoint to find $P(v)$, the ray tracing process is vectorized as a lookup table of cell indices for a virtual sensor. To decrease the cardinality of the search space, the vectorized ray tracing process is performed for all cells in a spherical field of view, and each cell's contribution to view quality is added to the quality of orientations for which the cell would be in the real sensor's field of view. This provides a fast way to find view quality for any orientation at a given location, and thereby decreases the search space from full 6D down to 3D, an exponential decrease in search space.

2.4 An Adaptable Object Reconstruction Algorithm

Algorithm 1 presents the architecture of the Adaptable Probabilistic Object Reconstruction Algorithm (APORA). This NBV-based algorithm enables mobile robots equipped with 3D sensors to reconstruct *a priori* unknown objects. The algorithm dynamically generates a search space of viewpoint candidates based on current knowledge of the unknown object. A view quality function calculates the expected information gain that could be obtained from each candidate view in the search space to select the next desired sensor pose. A navigation system performs dynamic path planning and obstacle avoidance to direct the robot to the NBV. During navigation, the volumetric map is constantly updated and a new NBV is computed once the current NBV has been reached or is found to be unreachable. This process is repeated until a termination criterion is met. The four required components of the algorithm are listed below:

Sensor Processing: The low-level sensor processing tasks: collecting point clouds from the 3D sensor, updating the volumetric map with the data, and de-

termining the robot pose (e.g. RGB-D SLAM).

View Quality Function: The probabilistic cost function discussed in Section III. It selects the viewpoint with highest expected information gain as the NBV.

Navigation: Path planning, obstacle avoidance, and robot locomotion. Path planning is dynamically updated as more of the environment is discovered, and determines if a NBV becomes unreachable as new parts of the object are observed.

Control Hub: The central control of autonomous behavior. This component uses the current volumetric map of the object to dynamically generate a search space of candidate viewpoints for the View Quality Function, determines when to compute new NBVs, and evaluates the termination criterion.

Algorithm 1 Adaptable, Probabilistic Object Reconstruction Algorithm

Require: Initial map $\mathcal{M}^0$
1: $k \leftarrow 0$
2: **while** termination criteria not met **do**
3: $\mathcal{S} \leftarrow$ Generate NBV search space
4: $\mathcal{X}^{k+1} \leftarrow \arg\max_{\mathcal{X} \in \mathcal{S}} Q(\mathcal{X}, \mathcal{M}^k)$
5: **while** $\mathcal{X}^{k+1}$ not reached && $\mathcal{X}^{k+1}$ reachable **do**
6: Navigate towards $\mathcal{X}^{k+1}$
7: $\mathcal{M}^{k+1} \leftarrow$ Update with sensor measurement
8: **end while**
9: Evaluate termination criteria
10: $k \leftarrow k + 1$
11: **end while**

2.4.1 Search Space Generation

The proposed algorithm introduces a novel method of dynamic NBV searchspace generation that enables the algorithm to adapt to objects of any size. Existing volumetric reconstruction approaches such as Vasquez-Gomez et al., and Isler et

al. use pre-designed sets of candidate viewpoints that are assumed to cover the entirety of the object. These viewpoints are sampled from basic geometries such as spheres or cylinders. [56] [21] [22]. The proposed algorithm generates a set of candidate viewpoints based on the current representation of the object. As discussed in Section III, the volumetric map is first processed to determine all voxels that have a non-negligible probability of belonging to the object $P(o_i)$. Then a bounding box is chosen that contains all voxels within the sensor's range of any voxel with a non-negligible $P(o_i)$. This bounding box is gridded at a user-defined resolution (or a user-defined search space size) and the centers of grid cells are taken for 3D locations of viewpoints. This search space is then pruned according to any limitations on the sensor configuration space (e.g. maximum reachable sensor height for a mobile robot). As discussed in Section III, the View Quality Function efficiently calculates information gain for many sensor poses at each 3D candidate location that cover all directions of view. Thus, the size of the candidate search space is dynamically scaled with the size of the expected object, minimizing the required search time for small objects while having the ability to adapt to large objects.

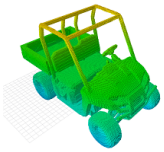
2.4.2 Termination Criterion

A simple termination criterion is used to determine completion of object reconstruction. The termination criterion is met if

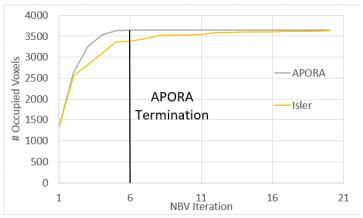
$$\mathcal{Q}(\mathcal{X}^{k+1}, \mathcal{M}^k) < q_{thresh} \tag{2.7}$$

I.e. when the information gain of the highest scoring candidate viewpoint (the

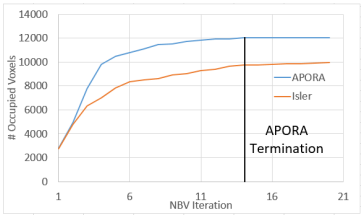
(a) Bunny (b) Volumetric Map (c) Robot (d) Volumetric Map (e) Vehicle



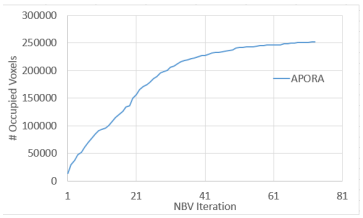
(f) Volumetric Map



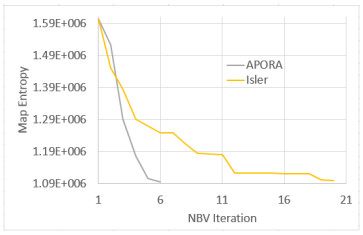
(g) Number of occupied voxels for bunny



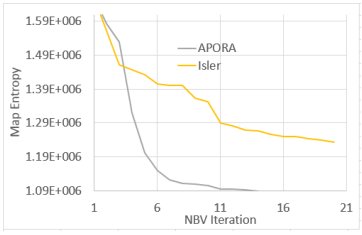
(h) Number of occupied voxels for robot



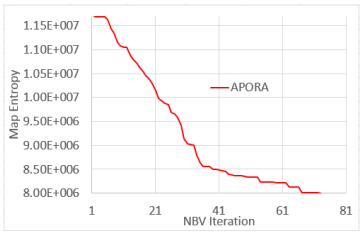
(i) Number of occupied voxels for vehicle



(j) Entropy in map (bunny)



(k) Entropy in map (robot)



(l) Entropy in map (vehicle)

Figure 2.3: Reconstruction of objects in simulation. Our algorithm is compared to the top-performing (Proximity Count) algorithm from Isler for the Stanford Bunny and Pioneer robot objects. [22]

NBV) falls below a user-defined threshold. This is the same criterion proposed by [22]. An advantage of this type of criterion is that it takes into account any restrictions on sensor configuration space. For example, if part of the object cannot be observed by any reachable candidate viewpoints, the algorithm will still terminate once no more information can be obtained from reachable viewpoints.

2.5 Simulation Results

To evaluate the performance of the proposed reconstruction algorithm and compare it to the state of the art, a simulation-based validation study was conducted using Gazebo[1] and ROS[2]. A simulated Microsoft Kinect RGB-D sensor with a maximum range of 1.5 meters was used, and iteratively placed at each NBV to view and reconstruct the object. The same APORA algorithm was used to reconstruct each of the three objects; no parameters were changed to make the algorithm work with the different sizes and complexities of the different objects. Since a "teleporting" navigation method was used for simulation, a simple collision detection method was used to check the validity of candidate viewpoints before moving the sensor. Note that on a real robot (e.g. the experiment in Section VI), the navigation system checks the validity of the NBV as the robot attempts navigation to the NBV.

Figure 2.3 shows the objects reconstructed and the results. Both algorithms used octree representations of the environment with 0.01 meter resolution. First, the Stanford Bunny[3] model was reconstructed using APORA as well as a reconstruction algorithm presented by Isler et al. [22] using the top-performing "proximity count" IG formulation. Isler released source code for the reconstruction algorithm, which was used to run these comparisons. The Stanford Bunny model was used in Isler's paper to compare several proposed IG formulations with algorithms by Kriegel et al. [29] and Vasquez-Gomez et al. [55]. Isler's results demonstrated that their algorithm performed comparably or better than those by Kriegel and Vasquez-Gomez. As may be observed from Figure 2.3g, both algorithms ended

[1] http://www.gazebosim.org
[2] http://www.ros.org
[3] Available from the Stanford University Computer Graphics Lab

	Isler	APORA
Average Number of Viewpoints	88	1.5×10^6
Average Time	8 s	12 s
Average Views/Second	11	1.3×10^6

Table 2.1: Computational speed results for robot object

up with complete volumetric reconstructions of the Bunny and therefore the same number of occupied voxels, but APORA converged to a complete model in significantly fewer NBV iterations (5 for APORA compared to 19 for Isler). As Figure 2.3j illustrates, the map entropy was also decreased much faster for APORA. As defined in Isler, map entropy is defined as the sum of entropies of voxels within a cube of 1.28 m side length surrounding the object. Note that the results differ somewhat from the results in Isler's paper because a different sensor with better coverage density was used (Microsoft Kinect vs. Stereo Cam).

Next, a Pioneer robot was used as the object to be reconstructed. This object, while of similar size to the bunny, is more complex and contains regions that are more difficult to view (e.g. between wheels and chassis and under the chassis). For this object, APORA significantly outperformed Isler. One reason for this improvement is the fact that APORA generates a vastly larger search space of candidate viewpoints and thus can find more precise views that observe more difficult regions of the object. Its computational efficiency allows it to search this larger space in a similar amount of time as Isler's algorithm. Isler's algorithm requires a predesigned set of viewpoints to be provided; we used a set of 88 views sampled from a dome centered over the object that was provided with the source code. Table 2.1 summarizes the computational time vs. search space for each algorithm. As can be seen, APORA evaluates viewpoints on the order of 1.0×10^5 faster than Isler.

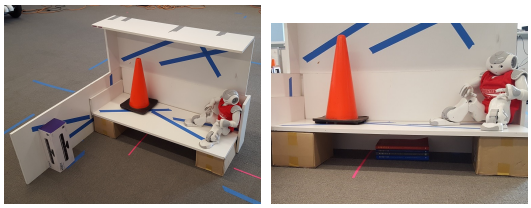


Figure 2.4: Object used in the experiment.

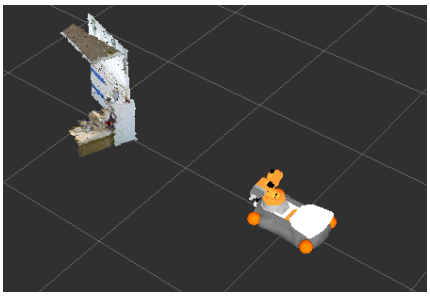
Finally, a vehicle was reconstructed by APORA using exactly the same algorithm setup to demonstrate its ability to adapt to larger objects. On the other hand, the setup for Isler's algorithm would need to be changed to enable it to reconstruct an object of that size. In addition, the large number of candidate viewpoints required to cover the entire vehicle would make Isler's algorithm very slow in computing NBVs. APORA had an average calculation time of 62 seconds for the vehicle reconstruction. This time could be brought down significantly by decreasing the resolution of the volumetric map or making the searchspace of viewpoints more coarse.

2.6 Experiment Results

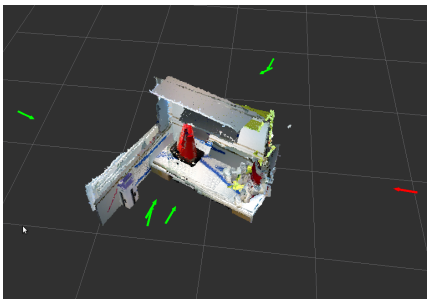
To demonstrate the capability of APORA on an actual robot in real time, we implemented it on a KUKA youBot equipped with an ASUS Xtion Pro Live (RGB-D sensor). Pose was provided using a combination of robot odometry and RTAB-Map [30] (RGB-D SLAM). RTAB-Map was also used to generate the map used for navigation and the point cloud of the object. The algorithm was implemented in ROS. Computation was split across three machines to maximize speed. The youBot computer ran the robot drivers, a laptop on the youBot ran sensor processing and RGB-D SLAM software, and an Intel Core i7 laptop connected over

WiFi ran the NBV search algorithm. Figure 2.4 shows the object used for the experiment. Note there are several complex surfaces on the object that can only be observed from special viewing positions: a high overhang, a low shelf with books underneath, and a traffic cone partially hidden by the walls. Colored tape was added to the scene to aid the RGB-D SLAM software. The reconstruction algorithm was run three times from different initial positions around the object. For each run, a small portion of the object was visible from the robot's starting position to seed the search algorithm. Figure 2.5a-2.5d shows the initial and final point cloud representation of the object in the first run of the experiment, as well as the volumetric representation of the object. Note that the difficult areas such as under the shelf were observed and reconstructed. The other two runs also achieved complete representations of the object, as shown in Figure 2.5e - 2.5h.

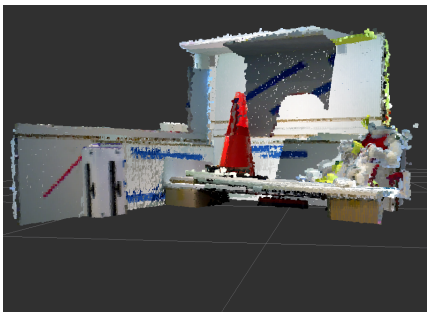
The robot took about 3:30 to finish reconstructing the object, showing that the algorithm can be run in real time. One run took 10 NBVs to converge, and the other two took 9 NBVs each. Regardless of the initial starting location, the robot had a quite efficient path for reconstructing the object, without needing to make several trips around the object. Due to error from the pose provider, there was significant noise in the reconstructed objects, which kept the algorithm from being able to distinguish or observe the small details in the object, such as the small occlusions behind the traffic cone. However even in the presence of noise it still did an excellent job finding all the major details of the object. A video of the reconstruction process is available accompanying this submission. The video demonstrates the robot selecting and navigating to NBVs while reconstructing the object.



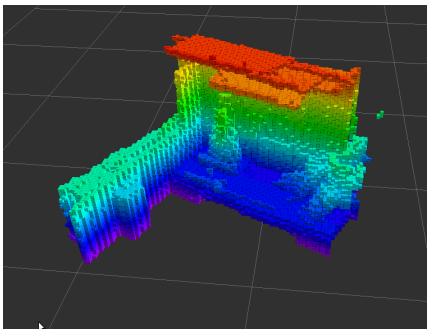
(a) Initial view of object



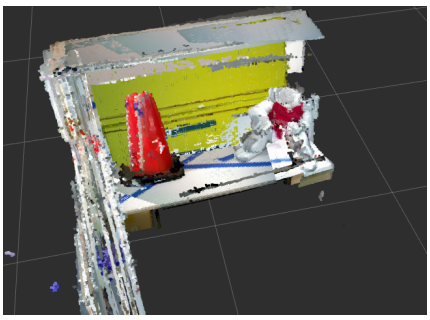
(b) Final object reconstruction.



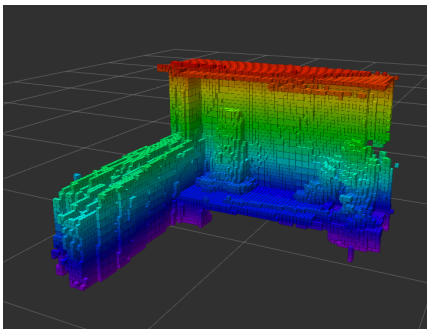
(c) Final object reconstruction.



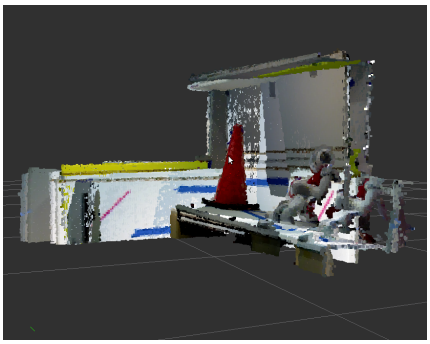
(d) Volumetric reconstruction of object.



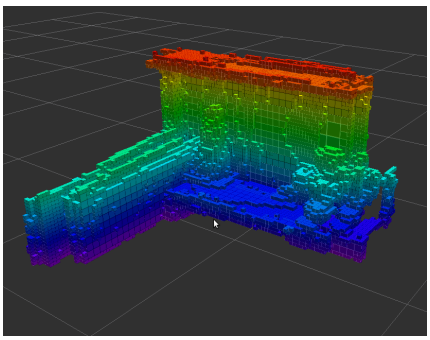
(e) Point cloud representation of 2^{nd} run



(f) Volumetric reconstruction of 2^{nd} run



(g) Point cloud representation of 3^{rd} run



(h) Volumetric reconstruction of 3^{rd} run

Figure 2.5: Results of autonomous reconfiguration system on KUKA youBot. Red arrow shows initial pose. Green arrows show NBV's.

2.7 Conclusion

A novel adaptable, probabilistic object reconstruction algorithm for mobile robots is presented, including a novel NBV cost formulation. While there are other NBV algorithms in literature, the proposed work is the first approach that enables mobile robots to explore and reconstruct an unknown object without making any assumptions on the size of the object. The algorithm uses an information-theoretic cost function that predicts the total amount of information gain that can be obtained from a candidate viewpoint, using a probabilistic framework to estimate potential information gain in unknown areas of the environment. The framework enables the algorithm to also consider self-occlusions by the object. Experiments in simulation and the real world confirm that the proposed algorithm can converge to a complete representation of an unknown object in a relatively small number of iterations, including complicated non-convex objects. It performs comparably or better than existing reconstruction algorithms on small objects, while being able to adapt to larger objects that cannot be reconstructed by existing algorithms. In addition, the information-based cost function allows the algorithm to have a natural termination criterion that indicates when the representation of the object is complete.

CHAPTER 3

AN INTEGRATED SYSTEM FOR PERCEPTION-DRIVEN AUTONOMY WITH MODULAR ROBOTS[1]

3.1 Introduction

Modular self-reconfigurable robot (MSRR) systems are composed of repeated robot elements (called *modules*) that connect together to form larger robotic structures, and can *self-reconfigure*, changing the connective arrangement of their own modules to form different structures with different capabilities. Since the field was in its nascence, researchers have presented a vision that promised flexible, reactive systems capable of operating in unknown environments. Modular self-reconfigurable robots would be able to enter unknown environments, assess their surroundings, and self-reconfigure to take on a form suitable to the task and environment at hand [63]. Today, this vision remains a major motivator for work in the field [64].

Continued research in MSRR has resulted in substantial advancement. Existing research has demonstrated MSRR self-reconfiguring, assuming interesting morphologies, and exhibiting various forms of locomotion, as well as methods for programming, controlling, and simulating modular robots [63, 66, 47, 36, 42, 23, ?, ?, ?, ?, ?, ?, 46, ?]. However, achieving autonomous operation of a self-reconfigurable robot in unknown environments requires a system with the ability to explore, gather information about the environment, consider the requirements of a high-level task, select configurations whose capabilities match the requirements of task and environment, transform, and perform actions (like manipulating ob-

[1]THIS WORK WAS CONDUCTED IN COLLABORATION WITH GANGYUAN JING FROM CORNELL UNIVERSITY AND TARIK TOSUN FROM UNIVERSITY OF PENNSYLVANIA

jects) to complete tasks. Existing systems provide partial sets of these capabilities. Many systems have demonstrated limited autonomy, relying on beacons for mapping [14, 10] and human input for high-level decision making [35, 9]. Others have demonstrated swarm self-assembly to address basic tasks like hill-climbing and gap-crossing [15, 39]. While these existing systems all represent advancements, none have demonstrated fully autonomous, reactive self-reconfiguration to address high-level tasks.

This paper presents a novel system allowing modular robots to complete complex high-level tasks autonomously. The system automatically selects appropriate behaviors to meet the requirements of the task and constraints of the perceived environment. Whenever the task and environment require a particular capability, the robot autonomously self-reconfigures to a configuration that has that capability. The success of this system is a product of our choice of system architecture, which balances distributed and centralized elements. Distributed, homogeneous robot modules provide flexibility, reconfiguring between morphologies to access a range of functionality. Centralized sensing, perception, and high-level mission planning components provide autonomy and decision-making capabilities. Tight integration between the distributed low-level and centralized high-level elements allows us to leverage advantages of distributed and centralized architectures.

The system is validated in three hardware demonstrations, showing that, based on a high-level task specification, the modular robot autonomously explores an unknown environment, decides if, when, and how to reconfigure, and manipulates objects to complete its task. By providing a clear example of how a modular robot system can be designed to leverage reactive reconfigurability in unknown environments, we have begun to lay the groundwork for reconfigurable systems to

address tasks in the real world.

3.2 Results

We demonstrate an autonomous, perception-informed, modular robot system that can reactively adapt to unknown environments via reconfiguration in order to perform complex tasks. The system hardware consists of a set of **robot modules** (that can move independently and dock with each other to form larger morphologies), and a **sensor module** that contains multiple cameras and a small computer for collecting and processing data from the environment. Software components consist of a **high-level planner** to direct robot actions and reconfiguration, and **perception algorithms** to perform mapping, navigation, and classification of the environment. Our implementation is built around the SMORES-EP modular robot [53], but could be adapted to work with other modular robots.

Our system is the first to demonstrate high-level decision-making in conjunction with reconfiguration in an autonomous setting. In three hardware demonstrations, the robot explores an *a priori* unknown environment, and acts autonomously to complete a complex task. Tasks are specified at a high level: users do not explicitly specify which configurations and behaviors the robot should use; rather, tasks are specified in terms of *behavior properties*, which describe desired effects and outcomes [24]. During task execution, the high-level planner gathers information about the environment and reactively selects appropriate behaviors from a design library, fulfilling the requirements of the task while respecting the constraints of the environment. Different configurations of the robot have different capabilities (sets of behaviors). Whenever the high-level planner recognizes that task and

environment require a behavior the current robot configuration cannot execute, it directs the robot to reconfigure to a different configuration that can execute the behavior.

Figure 5.7 shows the environments used for each demonstration, and Figure 3.2 shows snapshots during each of the demonstrations. A video of all three demonstrations is available as part of the supplementary material.

In Demonstration I, the robot must find, retrieve, and deliver all pink- and green-colored metal garbage to a designated drop-off zone for recycling, which is marked with a blue square on the wall. The demonstration environment contains two objects to be retrieved: a green soda can in an unobstructed area, and a pink spool of wire in a narrow gap between two trash cans. Various obstacles are placed in the environment to restrict navigation. When performing the task, the robot first explores using the "Car" configuration. Once it locates the pink object, it recognizes the surrounding environment as a "tunnel" type, and the high-level planner reactively directs the robot to reconfigure to the "Proboscis" configuration, which is then used to reach in between the trash cans and pull the object out in the open. The robot then reconfigures to the "Car," retrieves the object, and delivers it to the drop-off zone which the system had previously seen and marked during exploration. Figure 3.1b shows the resulting 3D map created from SLAM during the demonstration.

For Demonstrations II and III, the high-level task specification is the following: start with an object, explore until finding a delivery location, and deliver the object there. Each demonstration uses a different environment. For Demonstration II, the robot must place a circuit board in a mailbox (marked with pink-colored tape) at the top of a set of stairs with other obstacles in the environment. For

Demonstration III, the robot must place a postage stamp high up on the box that is sitting in the open.

For Demonstration II, the robot begins exploring in the "Scorpion" configuration. Shortly, the robot observes and recognizes the mailbox, and characterizes the surrounding environment as "stairs." Based on this characterization, the high-level planner directs the robot to use the "Snake" configuration to traverse the stairs. Using the 3D map and characterization of the environment surrounding the mail bin, the robot navigates to a point directly in front of the stairs, faces the bin, and reconfigures to the "Snake" configuration. The robot then executes the stair climbing gait to reach the mail bin, and drops the circuit successfully. It then descends the stairs and reconfigures back to the "Scorpion" configuration to end the mission.

For Demonstration III, the robot begins in the "Car" configuration, and cannot see the package from its starting location. After a short period of exploration, the robot identifies the pink square marking the package. The pink square is unobstructed, but is approximately 25cm above the ground; the system correctly characterizes this as the "high"-type environment, and recognizes that reconfiguration will be needed to reach up and place the stamp on the target. The robot navigates to a position directly in front of the package, reconfigures to the "Proboscis" configuration, and executes the "highReach" behavior to place the stamp on the target, completing its task.

3.3 Discussion

Modular self-reconfigurable robots are by their nature mechanically distributed, and as a result lend themselves naturally to distributed planning, sensing, and control. Most past systems have used entirely distributed frameworks [66, 47, 36, 10, 35, 39]. Our system is designed differently. It is distributed at the low level (hardware), but centralized at the high level (planning and perception), leveraging the advantages of both design paradigms.

The three scenarios in the demonstrations showcase a range of different ways SMORES-EP can interact with environments and objects: movement over flat ground, fitting into tight spaces, reaching up high, climbing over rough terrain, and manipulating objects. This broad range of functionality is only accessible to SMORES-EP by reconfiguring between different morphologies.

The high-level planner, environment characterization tools, and library work together to allow tasks to be represented in a flexible and reactive manner. For example, at the high level, Demonstrations II and III are the same task: deliver an object at a point of interest. However, after characterizing the different environments ("High" in II, "Stairs" in III), the system automatically determines that different configurations and behaviors are required to complete each task: the Proboscis to reach up high, and the Snake to climb the stairs. Similarly, in Demonstration I there is no high-level distinction between the green and pink objects - the robot is simply asked to retrieve all objects it finds. The sensed environment once again dictates the choice of behavior: the simple problem (object in the open) is solved in a simple way (with the Car configuration), and the more difficult problem (object in tunnel) is solved in a more sophisticated way (by reconfiguring into the Proboscis). Achieving this level of sophistication in control and decision-making

through a distributed architecture would have been significantly more difficult.

Centralized sensing and control during reconfiguration, provided by AprilTags and a centralized path planner, allowed our implementation to transform between configurations more rapidly than previous distributed systems. Each reconfiguration action (a module disconnecting, moving, and reattaching) takes about one minute. In contrast, past systems that utilized distributed sensing and control required 5-15 minutes for single reconfiguration actions [66, 47, 36], which would prohibit their use in the complex tasks and environments that our system demonstrated.

3.3.1 Challenges and Limitations

Through the hardware demonstrations performed with our system, we observed several challenges and opportunities for future improvement with autonomous perception-informed modular systems. All SMORES-EP body modules are identical, and therefore interchangeable for the purposes of reconfiguration. However, the sensor module has a significantly different shape than a SMORES-EP body module, which introduces heterogeneity in a way that complicates motion planning and reconfiguration planning. Configurations and behaviors must be designed to provide the sensor module with an adequate view, and to support its weight and elongated shape. Centralizing sensing also limits reconfiguration: modules can only drive independently in the vicinity of the sensor module, preventing the robot from operating as multiple disparate clusters.

Our high-level planner assumes all underlying components are reliable and robust, so failure of a low-level component can cause the high-level planner to behave

unexpectedly, and result in failure of the entire task. Table 4.1 shows the causes of failure for 24 attempts of Demonstration II (placing the stamp on the package). Nearly all failures are due to an error in one of the low-level components the system relies upon, with 42% of failure due to hardware errors and 38% due to failures in low-level software (object recognition, navigation, environment characterization). This kind of cascading failure is a weakness of centralized, hierarchical systems: distributed systems are often designed so that failure of a single unit can be compensated for by other units, and does not result in global failure.

This lack of robustness represents a challenge, but steps can be taken to address it. Unsurprisingly, open-loop behaviors (like stair-climbing and reaching up to place the stamp) were the least robust, and vulnerable to small hardware errors. Closing the loop using sensing made exploration and reconfiguration significantly less vulnerable to error. Future systems could be made more robust by introducing more feedback from low-level components to high-level decisions making processes, and by incorporating existing high-level failure-recovery frameworks [34]. Distributed repair strategies could also be explored, to replace malfunctioning modules with nearby working ones on the fly [**?**].

To implement our perception characterization component, we assumed a simplified set of environment types and implemented a simple characterization function to distinguish between them. This function does not generalize very well to completely unstructured environments and also is not very scalable. Thus, to expand the system to work well for more realistic environments and to distinguish between a large number of environment types, a more general characterization function should be implemented.

This paper presents the first modular robot system to autonomously complete

high-level tasks by reactively reconfiguring in response to its perceived environment and task requirements. In addition, putting the entire system to the test in hardware demonstrations revealed several opportunities for future improvement in such systems.

3.4 Methods and Materials

The following sections discuss the role of each component within the general system architecture. Inter-process communication between the many software components in our implementation is provided by the Robot Operating System (ROS)[2]. Figure 4.7 gives a flowchart of the entire system. For more details of the implementation used in the demonstrations see the Supplementary Materials.

3.4.1 Hardware

SMORES-EP Modular Robot: Each SMORES-EP module is the size of an 80mm cube and has four actuated joints, including two wheels that can be used for differential drive on flat ground [53], [54]. The modules are equipped with electro-permanent (EP) magnets that allow any face of one module to connect to any face of another, allowing the robot to self-reconfigure. The magnetic faces can also be used to attach to objects made of ferromagnetic materials (e.g. steel). The EP magnets require very little energy to connect and disconnect, and no energy to maintain their attachment force of 90N [53].

Each module has an onboard battery, microcontroller, and WiFi module to send

[2]http://www.ros.org

and receive messages. In this work, clusters of SMORES-EP modules are controlled by a central computer running a Python program that sends WiFi commands to control the four DoF and magnets of each module. Wireless networking is provided by a standard off-the-shelf router, with a range of about 100 feet, and commands to a single module can be received at a rate of about 20hz. Battery life is about one hour (depending on motor, magnet, and radio usage).

Sensor Module: SMORES-EP modules have no sensors that allow them to gather information about their environment. To enable autonomous operation, we introduce a *sensor module*, shown in Figure 4.8. The sensor module used in our demonstrations was designed to work with SMORES-EP, and is shown in Figure 4.8. The body of the sensor module is a 90mm $\times$ 70mm $\times$ 70mm box with thin steel plates on its front and back that allow SMORES-EP modules to connect to it. Computation is provided by an UP computing board with an Intel Atom 1.92 GHz processor, 4 GB memory, and a 64 GB hard drive. A USB WiFi adapter provides network connectivity. A front-facing Orbecc Astra Mini camera provides RGB-D data, enabling the robot to explore and map its environment and recognize objects of interest. A thin stem extends 40cm above the body, supporting a downward-facing webcam. This camera provides a view of a 0.75m $\times$ 0.5m area in front of the sensor module, and is used to track AprilTag [40] fiducials for reconfiguration. A 7.4V, 2200mAh LiPo battery provides about one hour of running time.

A single sensor module carried by the cluster of SMORES-EP modules provides centralized sensing and computation. Centralizing sensing and computation in a single sensor module has the advantage of facilitating control, task-related decision making, and rapid reconfiguration, but has the disadvantage of introducing

physical heterogeneity that makes it more difficult to design configurations and behaviors. The shape of the sensor module can be altered by attaching lightweight passive cubes that provide structure modules can connect to. Cubes have the same 80mm form factor as SMORES-EP modules, with magnets on all faces for attachment.

3.4.2 Perception and Planning for Information

Completing tasks in unknown environments requires the robot to explore and gain information about its surroundings, and use that information to inform actions and reconfiguration. Our system architecture includes active perception components to perform SLAM, choose waypoints for exploration, and recognize objects and regions of interest. It is also includes a framework to characterize the environment in terms of robot configurations abilities, allowing the high-level planner to reactively reconfigure the robot to adapt to different environment types. Implementations of these tools should be selected to fit the MSRR system being used and types of environments expected to be encountered.

Environment characterization is done using a discrete classifier (using the 3D occupancy grid of the environment as input) to distinguish between a discrete set of environment types corresponding to the library of robot configurations and gaits. To implement our system for a particular MSRR, the classification function must be defined by the user to classify the desired types of environments. For our proof-of-concept hardware demonstrations, we assumed a simplified set of possible environment types around objects of interest. We assumed the object of interest must be in one of four environment types shown in Figure 3.5e: "tunnel" (the object is in a narrow corridor), "stairs" (the object is at the top of low

stairs), "high" (the object is on a wall above the ground), and "free" (the object is on the ground with no obstacles around). Our implemented function performs characterization as follows: When the system recognizes an object in the environment, the characterization function evaluates the 3D information in the object's surroundings. It creates an occupancy grid around the object location, and denotes all grid cells within a robot-radius of obstacles as unreachable (illustrated in Figure 4.2). The algorithm then selects the closest reachable point to the object within 20^o of the robot's line of sight to the object. If the distance from this point to the object is greater than a threshold value and the object is on the ground, the function characterizes the environment as a "tunnel". If above the ground, it function the environment as a "stairs" environment. If the closest reachable point is under the threshold value, the system assigns a "free" or "high" environment characterization, depending on the height of the colored object.

Based on the environment characterization and target location, the function also returns a waypoint for the robot to position itself to perform its task (or to reconfigure, if necessary). In Demonstration II, the environment characterization algorithm directs the robot to drive to a waypoint at the base of the stairs, which is the best place for the robot to reconfigure and begin climbing the stairs.

Our implementation for other components of the perception architecture use previous work and open-soure algorithms. The RGB-D SLAM software package RTAB-MAP[31] provides mapping and robot pose. The system incrementally builds a 3D map of the environment and stores the map in an efficient octree-based volumetric map using Octomap[20]. The Next Best View algorithm by Daudelin et. al.[5] enables the system to explore unknown environments by using the current volumetric map of the environment to estimate the next reachable sensor viewpoint

that will observe the largest volume of undiscovered portions of objects (the Next Best View). In the example object delivery task, the system begins the task by iteratively navigating to these Next Best View waypoints to explore objects in the environment until discovering the dropoff zone.

To identify objects of interest in the task (such as the dropoff zone), we implemented our system using color detection and tracking. The system recognizes colored objects using CMVision[3], and tracks them in 3D[4] using depth information from the onboard RGB-D sensor. Although we implement object recognition by color, more sophisticated methods could be used instead, under the same system architecture.

3.4.3 Library of Configurations and Behaviors

A library-based framework is employed to organize user-designed configurations and behaviors for the SMORES-EP robot. Users can create designs for modular robot using our simulation tool and save designs to a library. Configurations and behaviors are labeled with properties, which are high-level descriptions of behaviors. Specifically, environment properties specify the appropriate environment that the behavior is designed for (e.g. a 3 module-high ledge) and behavior properties specify the capabilities of the behavior (e.g. climb). Therefore in this framework, a library entry is defined as $l = (C, B_C, P_b, P_e)$ where C is a robot configuration, B_C is the behavior of C, P_b is a set of behavior properties, and P_e is a set of environment properties. The high-level planner then can select appropriate configurations and behaviors based on given task specifications and environment

[3]CMVision: http://www.cs.cmu.edu/∼jbruce/cmvision/
[4]Lucas Coelho Figueiredo: https://github.com/lucascoelho91/ballFollower

information from the perception subsystem to accomplish the robot tasks. In our Demonstration II, the task specifications require the robot to deliver an object to a mailbox and the environment characterization algorithm reports that the mailbox is in a "stairs"-type environment. Then the high-level planner search the design library for a configuration and behavior that is capable to climb stairs with the object. Each entry is capable of controlling the robot to perform some actions in a specific environment. In Demonstration II, we show a library entry which controls the robot to "climb" a "stairs"-type environment.

To aid users in designing configurations and behaviors, we created a design tool called VSPARC[5]) and made it available online [24]. Users can use VSPARC to create, simulate and test designs in various environment scenarios with an included physics engine. Moreover, users can save their designs of configurations (connectivity among modules) and behaviors (joint commands for each module) in plain text files on our server and share them with other users. More importantly, all behaviors designed in VSPARC can be used to directly control the SMORES-EP robot system to perform the same action. Table 3.1 lists 10 entries for four different configurations that are used in this work.

3.4.4 Reconfiguration

When the high-level planner decides to use a new configuration during a task, the robot must reconfigure. We have implemented tools for mobile reconfiguration with SMORES-EP, taking advantage of the fact that individual modules can drive on flat surfaces. As discussed in Section 4.3.2, a downward-facing camera on the Sensor Module provides a view of a 0.75m $\times$ 0.5m area on the ground in

[5]www.vsparc.org

front of the sensor module. Within this area, the localization system provides pose for any module equipped with an AprilTag marker to perform reconfiguration. Given an initial configuration and a goal configuration, the reconfiguration controller commands a set of modules to disconnect, move and reconnect to form the new topology of the goal configuration. Currently, reconfiguration plans from one configuration to another are created manually and stored in the library. However the framework can work with existing assembly planning algorithms ([59, 49]) to generate reconfiguration plans automatically. Figure 3.6 shows reconfiguration from the "Car" to the "Proboscis" during Demonstration 1.

3.4.5 High-Level Planner

In our architecture, the high-level planner subsystem provides a framework for users to specify robot tasks using a formal language, and generates a centralized controller that directs robot motion and actions based on environment information. Our implementation is based on the Linear Temporal Logic MissiOn Planning (LTLMoP) toolkit, which automatically generates robot controllers from user-specified high-level instructions using synthesis [13, **?**, **?**, **?**, **?**, **?**, **?**]

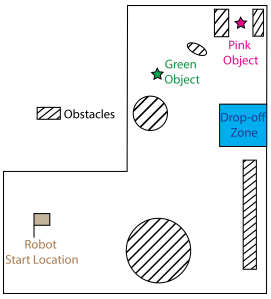
Consider the robot task in Demonstration II, the user indicates that the robot should **explore** until it locates the **mailBox**, then **drop** the object off. In addition, the user describes desired robot actions in terms of properties from the library. The high-level planner then generates a discrete robot controller that satisfies the given specifications. If no controller can be found or no appropriate library entries can implement the controller, users are advised to change the task specifications or add more behaviors to the design library.

The high-level planner coordinates each component of the system to control our MSRR to achieve complex tasks. At the system level, the sensing components gather and process environment information for the high-level planner, which then takes actions based on the given robot tasks by invoking appropriate low-level behaviors. In Demonstration II, when the robot is asked to deliver the object, the perception subsystem informs the robot that the mailbox is in a "stairs"-type environment. Therefore, the robot self-reconfigures to a "Snake" configuration to climb the stairs and deliver the object.

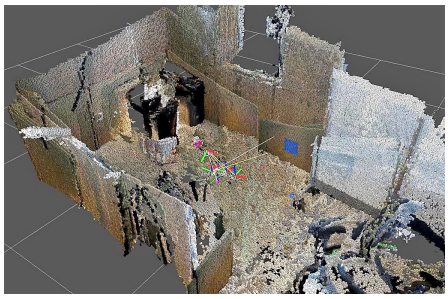
Acknowledgments

This work was funded by NSF grant numbers CNS-1329620 and CNS-1329692.

Figures and Tables



(a) Diagram of Demonstration I environment

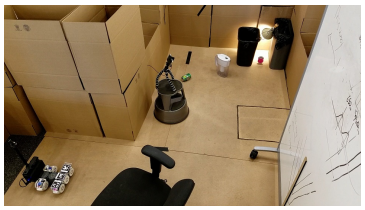


(b) Volumetric map of environment 1 built by visual SLAM

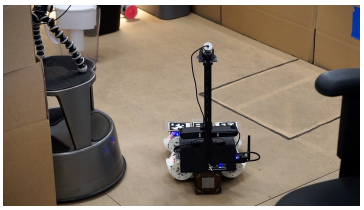
Environment Setup	Task Description
	**Demonstration I:** Explore environment to find all pink or green objects and blue dropoff zone. Deliver all objects to dropoff zone.
	**Demonstration II:** Explore environment to find mailbox, then deliver a circuit to the box.
	**Demonstration III:** Explore environment to find package, then place a stamp on the package.

(c) Environments and tasks for hardware demonstrations

Figure 3.1: Environments and Tasks for Demonstrations



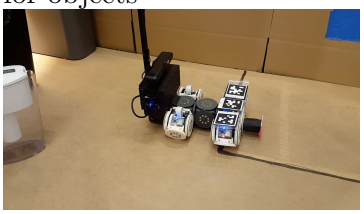
1. Environment and robot starting location



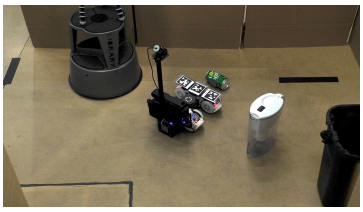
2. Exploring while searching for objects

3. Reconfiguring to retrieve pink object

4. Retrieving pink object

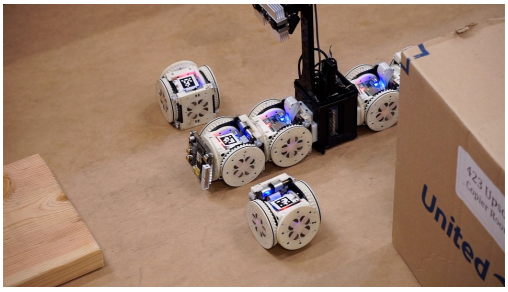


5. Depositing an object in the drop-off zone

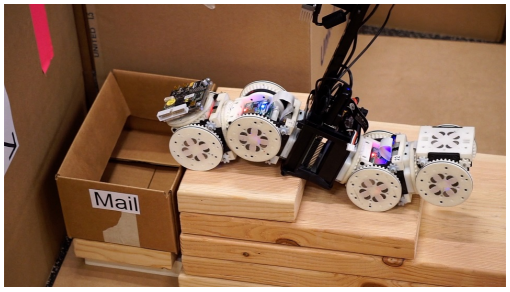


6. Retrieving green object

(a) Phases of Demonstration I.



1. Reconfiguring to climb stairs



2. Successful circuit delivery

1. Reconfiguring to place stamp

2. Successful stamp placement

(b) Demonstrations II and III.

Figure 3.2: Demonstrations 1, 2, and 3

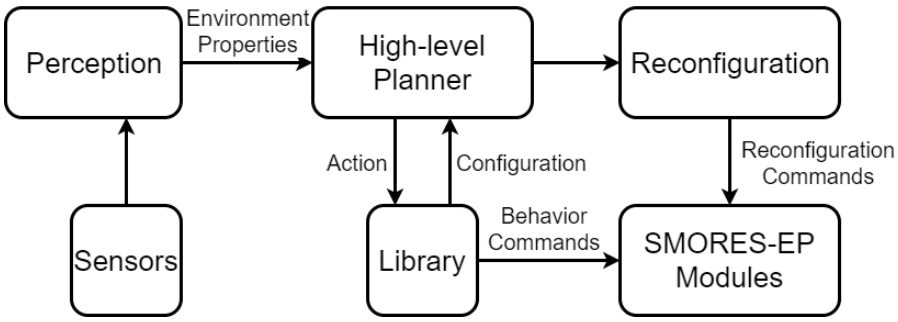
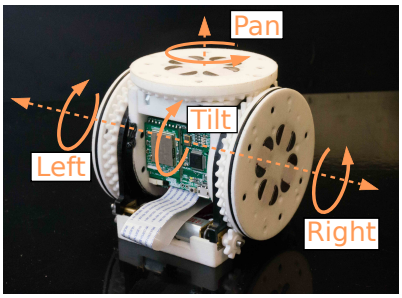


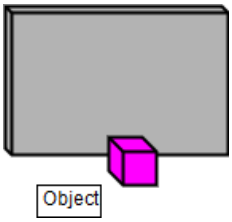
Figure 3.3: System Overview Flowchart



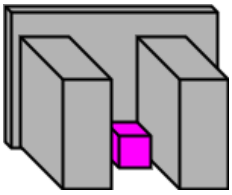
(a) SMORES-EP module

(b) Sensor Module with labelled components. UP board and battery are inside the body.

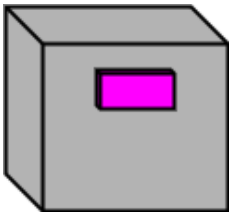
Figure 3.4: SMORES-EP Module and Sensor Module



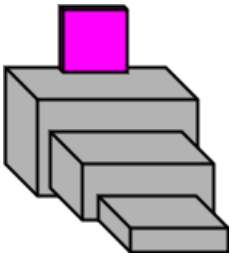
(a) **"free"** environment



(b) **"tunnel"** environment

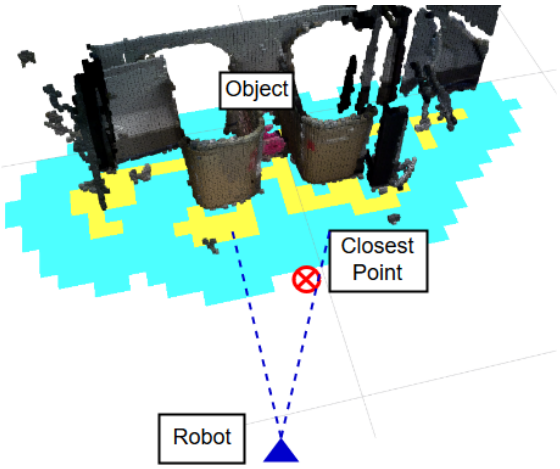


(c) **"high"** environment



(d) **"stairs"** environment

(e) Environment characterization types.



(f) An example of a **tunnel** environment characterization. Yellow grid cells are occupied, light blue cells are unreachable resulting from bloating obstacles.

Figure 3.5: Environment Characterization

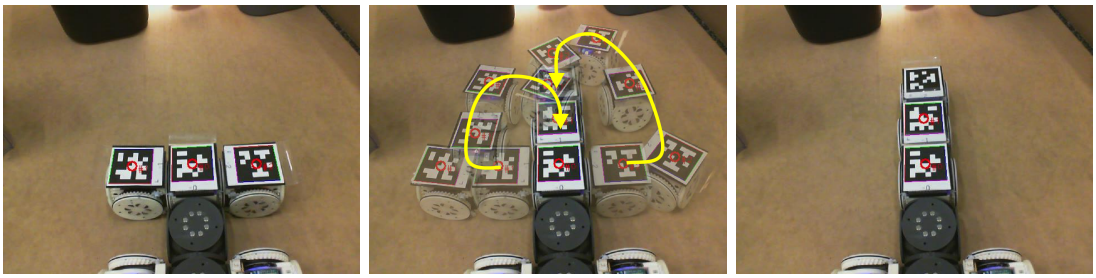
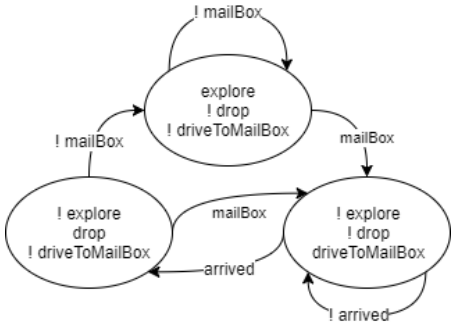


Figure 3.6: Module movement during reconfiguration. Left: initial configuration ("Car"). Middle: module movement, using AprilTags for localization. Right: final configuration ("Proboscis").

Specification 1 Drop an object to the mailbox

do **explore** if and only if the robot is not sensing **mailBox**
do **driveToMailBox** if and only if the robot is sensing (**mailBox** and not **arrived**)
do **drop** if and only if the robot is sensing (**arrived** and **mailBox**)

(a) Specification for dropping an object in the mailbox.



(b) The synthesized controller. A proposition with "!" has a value of `False`, and `True` otherwise.

Figure 3.7: A task specification with the synthesized controller.

Configuration	Behavior properties	Environment Types
Car	**pickUp**	"free"
	drop	"free"
	drive	"free"
Proboscis	**pickUp**	"tunnel" or "free"
	drop	"tunnel" or "free"
	highReach	"high"
Scorpion	**drive**	"free"
Snake	**climbUp**	"stairs"
	climbDown	"stairs"
	drop	"stairs" or "free"

Table 3.1: A library of robot behaviors

Reason of failure	**Number of times**	**Percentage**
Hardware Issues	10	41.7%
Navigation Failure	3	12.5%
Perception-Related Errors	6	25%
Network Issues	1	4.2%
Human Error	4	16.7%

Table 3.2: Reasons for demonstration failure.

Supplementary Materials

3.5 Additional Commentary on Related Work

Here we provide a more detailed overview of prior work in MSRR systems. These systems provide partial sets of the capabilities of our system.

The Millibot system demonstrated mapping when operating as a swarm. Certain members of the swarm are designated as "beacons," and have known locations. The autonomy of the Millibot swarm is limited: a human operator makes all high-level decisions, and is responsible for navigation using a GUI [14].

The Swarm-Bots system has been applied in exploration [10] and collective manipulation [35] scenarios. Like the Millibots, some members of the swarm act as "beacons" that are assumed to have known location during exploration. In a collective manipulation task, Swarm-Bots have limited autonomy, with a human operator specifying the location of the manipulation target and the global sequence of manipulation actions.

In [39], Swarm-Bots demonstrate swarm self-assembly to climb a hill. Robots exhibit phototaxis, with the goal of moving toward a light source. When robots detect the presence of a hill (using tilt sensors), they aggregate to form a random connected structure to collectively surmount the hill. A similar strategy is employed to cross holes in the ground. In each case, the swarm of robots is loaded with a single self-assembly controller specific to an *a priori* known obstacle type (hill or hole). The robots do not self-reconfigure between specific morphologies, but rather self-assemble, beginning as a disconnected swarm and coming together to form a random connected structure. In our work, a modular robot completes high-level tasks by autonomously self-reconfiguring between specific morphologies with different capabilities. Our system differentiates between several types of en-

vironments using RGB-D data, and may choose to use different morphologies to solve a given high-level task in different environments.

The swarmanoid project (successor to the swarm-bots), uses a heterogeneous swarm of ground and flying robots (called "hand-", "foot-", and "eye-" bots) to perform exploration and object retrieval tasks [9]. Robotic elements of the swarmanoid system connect and disconnect to complete the task, but the decision to take this action is not made autonomously by the robot in response to sensed environment conditions. While the location of the object to be retrieved is unknown, the method for retrieval is known and constant.

Self-reconfiguration has been demonstrated with several other modular robot systems. CKbot, Conro, and MTRAN have all demonstrated the ability to join disconnected clusters of modules together [66, 47, 36]. In order to align, Conro uses infra-red sensors on the docking faces of the modules, while CKBot and MTRAN use a separate sensor module on each cluster. In all cases, individual clusters locate and servo towards each other until they are close enough to dock. These experiments do not include any planning or sequencing of multiple reconfiguration actions in order to create a goal structure appropriate for a task. Additionally, modules are not individually mobile, and mobile clusters of modules are limited to slow crawling gaits. Consequently, reconfiguration is very time consuming, with a single connection requiring 5-15 minutes.

Other work has focused on reconfiguration planning. Paulos et al. present a system in which self-reconfigurable modular boats self-assemble into prescribed floating structures, such as a bridge [42]. Individual boat modules are able to move about the pool, allowing for rapid reconfiguration. In these experiments, the environment is known and external localization is provided by an overhead

AprilTag system.

MSRR systems have demonstrated the ability to accomplish low-level tasks such as various modes of locomotion [63]. Recent work includes a system which integrates many low-level capabilities of a MSRR system in a design library, and accomplishes high-level user-specified tasks by synthesizing elements of the library into a reactive state-machine [23]. This system demonstrates autonomy with respect to task-related decision making, but is designed to operate in a fully known environment with external sensing.

Our system goes beyond existing work by using self-reconfiguration capabilities of an MSRR system to take autonomy a step further. The system uses perception of the environment to inform the choice of robot configuration, allowing the robot to adapt its abilities to surmount challenges arising from *a priori* unknown features in the environment. Through hardware demonstrations, we show that autonomous self-reconfiguration allows our system to adapt to the environment to complete complex tasks.

3.6 Library of Configurations and Behaviors

In this work, we use the architecture introduced in [23]. We encode the full set of capabilities of the modular robot, such as driving and picking up items, in a library of robot configurations and behaviors. To create robot configurations and behaviors, users can utilize our simulator toolbox VSPARC (Verification, Simulation, Programming And Robot Construction [6]) presented in [23]. VSPARC allows users to design, simulate and test configurations and behaviors for the SMORES-EP

[6] www.vsparc.org

robot system.

Our implementation relies on a framework first presented in [23], which is summarized here. A library entry is defined as $l = (C, B_C, P_b, P_e)$ where:

- C is the robot *configuration*, specified by the number of modules and the connected structure of the modules.
- B_C is a *behavior* that C can perform. A behavior is a controller that specifies commands for robot joints to perform a specific movement.
- P_b is a set of *behavior properties* that describes what B_C does.
- P_e is a set of *environment types* that describe the environments in which this library entry is suitable.

To specify tasks at the high level, behavior properties P_b are used to describe desired robot actions without explicitly specifying a configuration or behavior. Environment types P_e specify the conditions under which a behavior can be used. This allows the high-level planner to match environment characterizations from the perception subsystem with configurations and behaviors that can perform the task in the current environment. In Demonstration II, when the environment characterization algorithm reports that the mailbox is located in a "stairs"-type environment, the high-level planner queries the library for configurations that can climb stairs. Since the library indicates that current configuration is only capable of driving on flat ground, the high-level planner opts to reconfigure to the stair-climber configuration, and executes its **climbUp** behavior.

In [23], all robot behaviors are *static* behaviors. That is, once users create a behavior in VSPARC, joint values for each module are fixed and cannot be

modified during behavior execution. Static behaviors, such as a car with a fixed turning radius, do not provide enough maneuverability for the robot to navigate around unknown environment. In this work, we expand the type of behaviors in the library by using *parametric* behaviors, which were first introduced in [24]. Parametric behaviors have joint commands that can be altered during run-time, and therefore allow a wider range of motions. For example, a parametric behavior for a car configuration can be a driving action with two parameters: turning angle and driving velocity. The system associates a parametric behavior with a program that generates values of joint commands based on environment information and current robot tasks. Based on the sensed environment, the perception and exploration subsystem (Section 3.4.2) can generate a collision-free path, which is used to calculate real-time velocity for the robot. The system then converts the robot velocity to joint values in parametric behaviors at run-time.

To provide an illustrative example, this paper discusses two configurations and their capabilities in detail. The "Car" configuration shown in Figure 3.2a-5 is capable of picking up and dropping objects in a "free" environment. In addition, the "Car" configuration can locomote on flat terrain. It uses a parametric differential drive behavior to convert a desired velocity vector into motor commands (**drive** in Table 3.1).

The "Proboscis" configuration shown in Figure 3.2a-4 has a long arm in front, and is suitable for reaching between obstacles in a narrow "tunnel" environment to grasp objects or reaching up in a "high" environment to drop items. However, the locomotion behaviors available for this configuration are limited to forward/backward motion, making it unsuitable for general navigation.

This library-based framework allows users to express desired robot actions in

an abstract way by specifying behavior properties. For example, if a task specifies that the robot should execute a behavior with the **drop** property, the system could choose to use either the Car or Proboscis configurations to perform the action, since both have behaviors with the **drop** property. The decision of which configuration to use is made during task execution, based on the sensed environment. For example, if the perception system reports that the environment is of type "tunnel", the Proboscis configuration will be used, because the library indicates that it can be used in "tunnel"-type environments while the Car cannot.

3.7 High-Level Planner

In order to generate controllers from high-level task specifications, we first abstract the robot and environment status as a set of Boolean propositions. In Demonstration II, the robot action **drop** is `True` if the robot is currently dropping an object in the mailbox (and `False` otherwise) and the environment proposition **mailBox** is `True` if the robot is currently sensing a mailbox (and `False` otherwise). Moreover the proposition **explore** encodes whether or not the robot is currently searching for the target, the mailbox in this case.

By using a library of robot configurations and behaviors as well as environment characterization tools, we can map these high-level abstraction to low-level sensing programs and robot controllers. As discussed in Section 3.6, the user specifies high-level robot actions in terms of behavior properties from the library. In Demonstration II, our system can choose to do a drop action by executing any behavior from the library which has the behavior property **drop**, and which also satisfies the current "stairs"-type environment. If the current robot configuration

cannot execute an appropriate behavior, the robot will reconfigure to a different configuration that can. In this way, the system autonomously chooses to implement **drop** appropriately in response to the sensed environment. Our system evaluates propositions related to the state of the environment using perception and environment characterization tools in Section 3.4.2. For example, users can map the proposition **mailBox** to the color tracking function in our perception subsystem, which assign the value `True` to **mailBox** if and only if the robot is currently seeing a mailbox with the onboard camera. The system treats propositions, such as **explore**, that require the robot to navigate in the workspace differently from the other simple robot actions, such as **drop**. In this example, users can map **explore** to behavior property **drive**, which represents a set of parametric behaviors as discussed in Section 3.6. In order to obtain joint values for behaviors at run-time, a path planner in the perception and planning subsystem (Section 3.4.2) takes into account the robot goal as well as the current environment information from the perception subsystem, and generates a collision-free path for the robot to follow. Our system then converts this path to joint values, which are used to execute the **drive** behaviors.

Our implementation employs the Linear Temporal Logic MissiOn Planning (LTLMoP) toolkit to automatically generate robot controllers from user-specified high-level instructions using synthesis [13, 26]. The user describes the desired robot tasks with high-level specifications over the set of abstracted robot and environment propositions that are mapped to behavior properties from the library. LTLMoP automatically converts the specification to logic formulas, which are then used to synthesize a robot controller that satisfies the given tasks (if one exists). The controller is in the form of a finite state automaton, as shown in Figure 3.7b. Each state specifies a set of high-level robot actions that need to be performed, and

transitions between states include a set of environment propositions. Note some of propositions are omitted in Figure 3.7b for clarity. Execution of the high-level controller begins at the predefined initial state in the finite state automaton. In each iteration, LTLMoP determines the values of all environment propositions by calling the corresponding sensing program. Then, LTLMoP chooses the next state in the finite state machine by taking the transition that matches the current value of all environment propositions. In the next state, for each robot proposition LTLMoP chooses a behavior from the design library which satisfies both the behavior properties and current environment type. For example, in Figure 3.7b we start in the top state and execute the **explore** program. If the robot senses a mailbox, the value of **mailBox** becomes `True` and therefore the next state is the bottom right state. We then stop the **explore** program and execute the **driveToMailBox** program. We introduce additional constraints to the original task specifications to guarantee that there exist behaviors in the library to implement the synthesized controller. Since self-reconfiguration is time-consuming, the controller chooses to execute the selected behavior using the current robot configuration whenever possible. If the current configuration cannot execute the behavior, the controller instructs the robot to reconfigure to one that can, and if multiple appropriate configurations are available, the controller selects one at random.

3.8 Reconfiguration

When the high-level planner decides to use a new configuration during a task, the robot must reconfigure. Our system architecture allows any method for reconfiguration, provided that the method requires no external sensing. SMORES-EP is capable of all three classes of modular self-reconfiguration (chain, lattice, and mo-

bile reconfiguration) [7, 65]. We have implemented tools for mobile reconfiguration with SMORES-EP, taking advantage of the fact that individual modules can drive on flat surfaces as described in Section 4.3.2.

Determining the relative positions of modules during mobile self-reconfiguration is an important challenge. In this work, the localization method is centralized, using a camera carried by the robot to track AprilTag fiducials mounted to individual modules. As discussed in Section 4.3.2, the camera provides a view of a $0.75\text{m} \times 0.5\text{m}$ area on the ground in front of the sensor module. Within this area, the localization system provides pose for any module equipped with an AprilTag marker to perform reconfiguration.

Given an initial configuration and a goal configuration, the reconfiguration controller commands a set of modules to disconnect, move and reconnect in order to form the new topology of the goal configuration. The robot first takes actions to establish the conditions needed for reconfiguration by confirming that the reconfiguration zone is a flat surface free of obstacles (other than the modules themselves). The robot then sets its joint angles so that all modules that need to detach have both of their wheels on the ground, ready to drive. Then the robot performs operations to change the topology of the cluster by detaching a module from the cluster, driving, and re-attaching at its new location in the goal configuration, as shown in Figure 3.6. Currently, reconfiguration plans from one configuration to another are created manually and stored in the library. However the framework can work with existing assembly planning algorithms ([59, 49]) to generate reconfiguration plans automatically. Because the reconfiguration zone is free of obstacles, the controller compute collision-free paths offline and store them as part of the reconfiguration plan. Once all module movement operations have completed and the goal topology

is formed, the robot sets its joints to appropriate angles for the goal configuration to continue performing desired behaviors.

We developed several techniques to ensure reliable connection and disconnection during reconfiguration. When a module disconnects from the cluster, the electro-permanent magnets on the connected faces are turned off. To guarantee a clean break of the magnetic connection, the disconnecting module bends its tilt joint up and down, mechanically separating itself from the cluster. During docking, accurate alignment is crucial to the strength of the magnetic connection [53]. For this reason, rather than driving directly to its final docking location, a module instead drives to a pre-docking waypoint directly in front of its docking location. At the waypoint, the module spins in place slowly until its heading is aligned with the dock point, and then drives in straight to attach. To guarantee a good connection, the module intentionally overdrives its dock point, pushing itself into the cluster while firing its magnets.

CHAPTER 4

PERCEPTION-INFORMED AUTONOMOUS ENVIRONMENT AUGMENTATION WITH MODULAR ROBOTS[1]

4.1 Introduction

Employing structures to accomplish tasks is a ubiquitous part of the human experience: to reach an object on a high shelf, we place a ladder near the shelf and climb it, and at a larger scale, we construct bridges across wide rivers to make them passable. The fields of collective construction robotics and modular robotics offer examples of systems that can construct and traverse structures out of robotic or passive elements [46, 43, 52, 37], and assembly planning algorithms that allow arbitrary structures to be built under a variety of conditions [49, 59]. This existing body of work provides excellent contributions regarding the generality and completeness of these methods: some algorithms are provably capable of generating assembly plans for arbitrary volumetric structures in 3D, and hardware systems have demonstrated the capability to construct a wide variety of structures.

Less work is available regarding ways that robots could deploy structures as a means of completing an extrinsic task, the way a person might use a ladder to reach a high object. In this paper, we present hardware, perception, and high-level planning tools that allow structure-building to be deployed by a modular robot to address high-level tasks.

Our work uses the SMORES-EP modular robot [53], and introduces novel pas-

[1]THIS WORK WAS CONDUCTED IN COLLABORATION WITH GANGYUAN JING FROM CORNELL UNIVERSITY AND TARIK TOSUN FROM UNIVERSITY OF PENNSYLVANIA

sive block and wedge modules that SMORES-EP can use to form ramps and bridges in its environment. Building structures allows SMORES-EP to surmount large obstacles that would otherwise be very difficult or impossible to traverse, and therefore expands the set of tasks the robot can perform. This addresses a common weakness of modular robot systems, which often struggle with obstacles much larger than a module.

We expand on an existing framework for selecting appropriate robot morphologies and behaviors to address high-level tasks [24]. In this work, the high-level planner not only decides when to reconfigure the robot, but also when to augment the environment by assembling a passive structure. To inform these decisions, we introduce a novel environment characterization algorithm that identifies candidate features where structures can be deployed to advantage. Together, these tools comprise a novel framework to automatically identify when, where, and how the robot can augment its environment with a passive structure to gain advantage in completing a high-level task.

We integrate our tools into an existing system for perception-driven autonomy with modular robots [6], and validate them in two hardware experiments. Based on a high-level specification, a modular robot reactively identifies inaccessible regions and autonomously deploys ramps and bridges to complete locomotion and manipulation tasks in realistic office environments.

4.2 Related Work

Our work complements the well-established field of collective robotic construction, which focuses on autonomous robot systems for building activity. While we use

a modular robot to create and place structures in the environment, our primary concern is not assembly planning or construction of the structure itself, but rather its appropriate placement in the environment to facilitate completion of an extrinsic high-level task.

Petersen et al. present Termes [43], a termite-inspired collective construction robot system that creates structures using blocks co-designed with a legged robot. Similarly, our augmentation modules are designed to be easily carried and traversed by SMORES-EP. Where the TERMES project focused on collective construction of a goal structure, we are less concerned with efficient building of the structure itself and more concerned with the application and placement of the structure in the larger environment as a means of facilitating a task unrelated to the structure itself.

Werfel et al. present algorithms for environmentally-adaptive construction that can build around obstacles in the environment [59]. A team of robots senses obstacles and builds around them, modifying the goal structure if needed to leave room for immovable obstacles. An algorithm to build enclosures around preexisting environment features is also presented. As with Termes, the goal is the structure itself; while the robots do respond to the environment, the structure is not built in response to an extrinsic high-level task.

Napp et al. present hardware and algorithms for building amorphous ramps in unstructured environments by depositing foam with a tracked mobile robot [38, 37]. Amorphous ramps are built in response to the environment to allow a small mobile robot to surmount large, irregularly shaped obstacles. Our work is similar in spirit, but places an emphasis on autonomy and high-level locomotion and manipulation tasks rather than construction.

Modular self-reconfigurable robot (MSRR) systems are comprised of simple repeated robot elements (called *modules*) that connect together to form larger robotic structures. These robots can *self-reconfigure*, rearranging their constituent modules to form different morphologies, and changing their abilities to match the needs of the task and environment [64]. Our work leverages recent systems that integrate the low-level capabilities of an MSRR system into a design library, accomplish high-level user-specified tasks by synthesizing library elements into a reactive state machine [24], and operate autonomously in unknown environments using perception tools for environment exploration and characterization [6].

Our work extends the SMORES-EP hardware system by introducing passive pieces that are manipulated and traversed by the modules. Terada and Murata [52], present a lattice-style modular system with two parts, structure modules and an assembler robot. Like many lattice-style modular systems, the assembler robot can only move on the structure modules, and not in an unstructured environment. Other lattice-style modular robot systems create structures out of the robots themselves. M-blocks [46] form 3D structures out of robot cubes which rotate over the structure. Paulos et al. present rectangular boat robots that self-assemble into floating structures, like a bridge [42].

Magnenat et al [33] present a system in which a mobile robot manipulates specially designed cubes to build functional structures. The robot explores an unknown environment, performing 2D SLAM and visually recognizing blocks and gaps in the ground. Blocks are pushed into gaps to create bridges to previously inaccessible areas. In a "real but contrived experimental design" [33], a robot is tasked with building a three-block tower, and autonomously uses two blocks to build a bridge to a region with three blocks, retrieving them to complete its

task. Where the Magnenat system is limited to manipulating blocks in a specifically designed environment, our work presents hardware, perception, and high-level planning tools that are more general, providing the ability to complete high-level tasks involving locomotion and manipulation in realistic human environments.

4.3 Approach

4.3.1 Environment Characterization

To successfully navigate its environment, a mobile robot must identify traversable areas. One simple method for wheeled robots is to select flat areas large enough for the robot to fit. However, MSRR systems can reconfigure to traverse a larger variety of terrains. The augmentation abilities we introduce extend MSRR navigation even further; the robot can build structures to traverse otherwise-impossible terrains. For autonomous operation, we need an algorithm to locate and label features in the environment that can be augmented. We present a probabilistic, template-based environment characterization algorithm that identifies augmentable features from a 2.5D elevation map of the robot's environment.

The characterization algorithm searches for a desired feature template $\mathcal{F}_n$ which identifies candidate locations in the environment where useful structures could be built. A template consists of a grid of likelihood functions $l_i(h)$ for $1 \leq i \leq M$ where M is the number of grid cells in the template, and h is a height value. The size of grid cells in the template is variable and need not correspond to the resolution of the map. In addition, features of different size can be searched for by changing the cell size of the template to change the scale. In our system

implementation, template parameters and likelihood functions are designed by hand to correspond to each structure in the system's structure library. However, future implementations could automatically generate these templates offline with an additional algorithm.

Figure 4.1 shows an example of a template used to characterize a "ledge" feature, consisting of Gaussian and logistic likelihood functions. Any closed-form likelihood function may be used for each grid cell, enabling templates to accommodate noisy data and variability in possible geometric shapes of the same feature. To determine if the feature exists at a candidate pose $\mathcal{X}$ in the map, a grid of height values is taken from the map corresponding to the template grid centered and oriented at the candidate pose, as illustrated in Figure 4.1. Then, the probability that each grid cell c_i belongs to the feature is evaluated using the cell's likelihood function from the template.

$$P(c_i \in \mathcal{F}_n) = l_i(h_i) \tag{4.1}$$

The likelihood of the feature existing at that location is calculated by finding the total probability that all grid cells belong to the feature. Making the approximate simplifying assumption that grid cells are independent, this probability is equivalent to taking the product over the feature likelihoods of all grid cells in the template:

$$P(\mathcal{X} \in \mathcal{F}_n) = \prod_{i=1:M} l_i(h_i) \tag{4.2}$$

The feature is determined to exist if the total probability is higher than a user-defined threshold, or $P(\mathcal{X} \in \mathcal{F}_n) > \alpha^M$, where α represents the minimum average

probability of each grid cell forming part of the feature. In our experiments we use $\alpha = 0.95$. This formulation normalizes the threshold with respect to the number of grid cells in the template.

To characterize an environment, the algorithm takes as inputs an elevation map of the environment and a list of feature templates. Before searching for features, the algorithm preprocesses the elevation map by segmenting it into flat, unobstructed regions that are traversable without augmentation. It then grids the map and exhaustively evaluates each candidate feature pose from the grid, using a grid of orientations for each 2D location. In addition to evaluation with the template, candidate poses are only valid if the ends of the feature connect two traversable regions from the preprocessing step, thereby having potential to extend the robot's reachable space. Once the search is complete, the algorithm returns a list of features found in the map, including their locations, orientations, and the two regions they link in the environment. Figure 4.2 shows an example of a characterized map. Each long red cell represents a detected "ledge-height-2" feature, with a corresponding small pink cell demonstrating the orientation of the feature (and the bottom of the ledge). Note that, in this example, several features are chosen close to each other. Since all connect the same regions, any one is valid and equivalent to be selected for augmentation.

The algorithm scales linearly with the number of grid cells in the 2D environment map, and linearly with the number of features being searched for. Characterization of the environment shown in Figure 4.2 took approximately 3 seconds to run on a laptop with an Intel Core i7 processor.

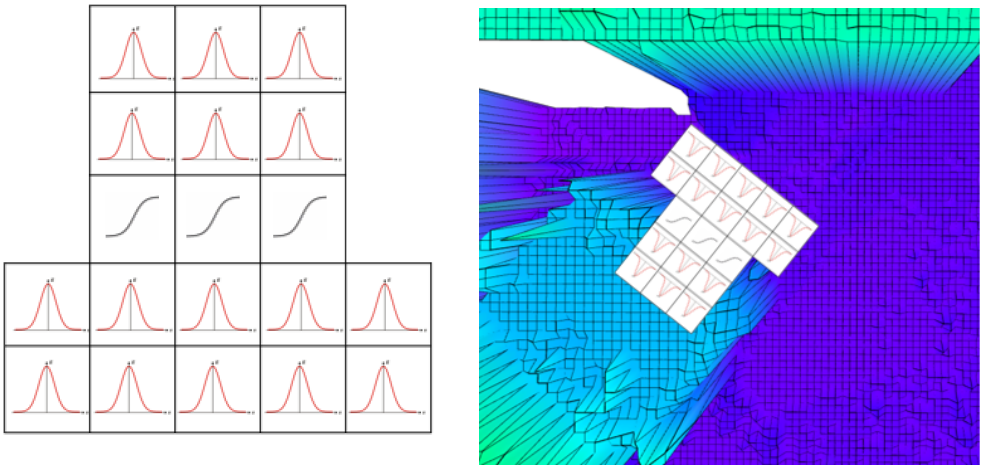


Figure 4.1: *Left:* Example template used to characterize a "ledge" feature. *Right:* Example template overlayed on elevation map (top view) to evaluate candidate feature pose.

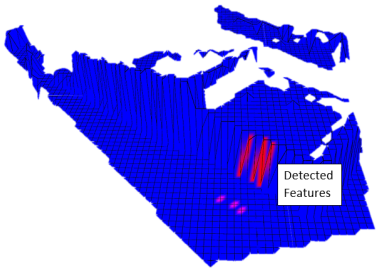


Figure 4.2: Characterization of an environment with a "ledge" feature. Red indicates a detected feature, pink indicates the start of the feature, demonstrating orientation.

4.3.2 Hardware: Augmentation Modules

Our system is built around the SMORES-EP modular robot. Each module is the size of an 80mm cube, weighs 473g, and has four actuated joints, including two wheels that can be used for differential drive on flat ground [53], [54]. Electropermanent (EP) magnets allow any face of one module to connect to any face of another, enabling the robot to self-reconfigure. They are also able to attach to objects made of ferromagnetic materials (e.g. steel). The EP magnets require very little energy to connect and disconnect, and no energy to maintain their attachment force of 90N [53]. Each module has its own battery, microcontroller, and WiFi

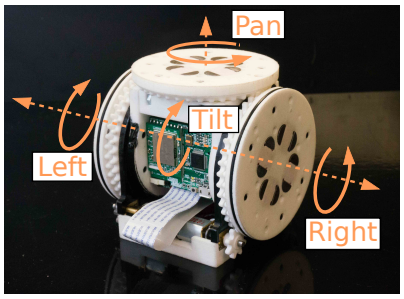


Figure 4.3: SMORES-EP module

module for communications. In this work, clusters of modules are controlled by a central computer running a Python program that commands movement and magnet via WiFi. Wireless networking is provided by a standard off-the-shelf router, and commands to a single module can be received at a rate of about 20hz. Battery life is about one hour (depending on magnet, motor, and radio usage).

Large obstacles, like tall ledges or wide gaps in the ground, are often problematic for modular robot systems. One might expect that a modular system could scale, addressing a large-length-scale task by using many modules to form a large robot. In reality, modular robots don't scale easily: adding more modules makes the robot bigger, but not stronger. The torque required to lift a long chain of modules grows quadratically with the number of modules, quickly overloading the maximum torque of the first module in the chain. Consequently, large systems become cumbersome, unable to move their own bodies. Simulated work in reconfiguration and motion planning has demonstrated algorithms that handle hundreds of modules, but in practice, fixed actuator strength has typically limited these robots to configurations with fewer than 40 modules.

We address this issue by extending the SMORES-EP hardware system with passive elements called *environment augmentation modules*. We use the *Wedge* and *Block* augmentation modules shown in Figure 4.4. Wedge and block modules are designed to work synergistically with SMORES-EP, providing features that use

the best modes of locomotion (driving), manipulation (magnetic attachment), and sensing (AprilTags) available to SMORES-EP.

Blocks are the same size as a module (80mm cube), and wedges are half the size of a block (an equilateral right triangle with two 80mm sides). Both are made of lightweight laser-cut medium-density fiberboard (blocks are 162g, wedges are 142g) and equipped with a steel attachment point for magnetic grasping. Neodymium magnets on the back faces of wedges, and the front and back faces of blocks, form a strong connection in the horizontal direction. Interlocking features on the top and bottom faces of the blocks, and the bottom faces of the wedges, allow them to be stacked vertically. Wedges provide a 45-degree incline with a high-friction rubber surface, allowing a set of 3 or more modules to drive up them. Side walls on both the wedges and blocks ensure that SMORES-EP modules stay aligned to the structure and cannot fall off while driving over it. The side walls of wedges are tapered to provide a funneling effect as modules drive onto them, making them more tolerant to misalignment. Each wedge and ramp has unique AprilTag fiducials on its faces, allowing easy identification and localization during construction and placement in the environment.

Wedges and blocks allow a SMORES-EP cluster to autonomously construct bridges or ramps that allow it to reach higher heights and cross wider gaps than it could with robot modules alone (Figure 4.5). Provided enough time, space, and augmentation modules are available, there is no limit to the height of a ramp that can be built. Bridges have a maximum length of 480mm (longer bridges cannot support a load of three SMORES-EP modules in the center).

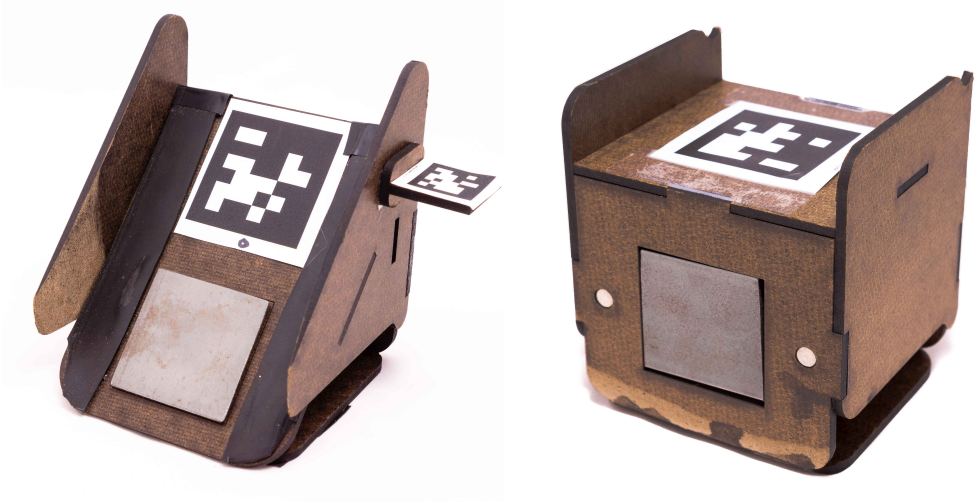


Figure 4.4: Wedge and Block Augmentation Modules

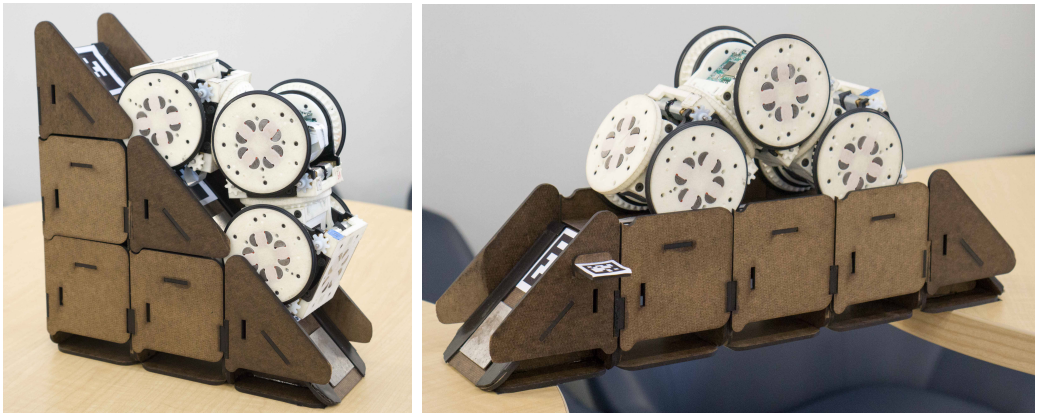


Figure 4.5: Bridge and Ramp

4.3.3 High-Level Planner

We utilize a high-level planner that allows users to control low-level robot actions by defining tasks at high-level with a formal language [6]. The high-level planner serves two main functions. First it acts as a mission planner, automatically *synthesizing* a robot controller (finite state automaton) from user-given task specifications. Second, it *executes* the generated controller, commanding the robot to react to the sensed environment and complete the tasks. In this work, the high-level

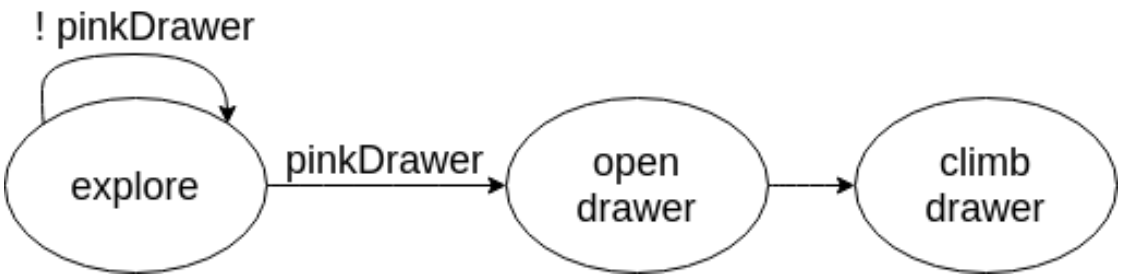


Figure 4.6: An example of synthesized robot controller

planner integrates with a *robot design library* of user-created robot configurations and behaviors, as well as a *structure library* of structures that can be deployed to alter the environment. Users do not explicitly specify configurations and behaviors for each task, but rather define goals and constraints for the robot. Based on the task specifications, the high-level planner chooses robot configurations and behaviors from the design library, and executes them to satisfy the tasks. When necessary, the planner will also choose to build a structure from the structure library to facilitate task execution.

Consider the following example task: The robot is asked to look for a pink drawer, open the drawer, and then climb on top of it. The mission planer synthesizes the controller shown in Figure 4.6. Each state in the controller is labeled with a desired robot action, and each transition is labeled with perceived environment information; for example, the "climb drawer" action is specified to be any behavior from the library with properties *climb* in a *ledge* environment. In our previous framework [6], the high-level planner could choose to reconfigure the robot whenever needed to satisfy the required properties of the current action and environment.

In this work, the high-level planner can choose not only to change the abilities of the robot (reconfiguration), but also the properties of the environment (envi-

ronment augmentation). We expand our framework by introducing a library of structures $S = \{s_1, s_2, \ldots, s_N\}$, where each structure is defined as $s_n = \{\mathcal{F}_n, A_n\}$. $\mathcal{F}_n$ is an environment feature template that specifies the kind of environment the structure can augment, and which can be identified by the environment characterization algorithm described in Section 4.3.1. The *assembly plan* A_n is itself a high-level task controller (finite state automaton), specifying the required building blocks needed to create the structure and the order in which they may be assembled. As with other tasks in our framework, construction actions within assembly plans are specified in terms of behavior properties (e.g. *pickUpBlock*, *placeWedge*) that the high-level planner maps to appropriate configurations and behaviors from the robot design library.

For the example in Figure 4.6, if no behavior in the library satisfies the "climb drawer" action, the high-level planner will consider augmenting its environment with a structure. It passes a set of feature templates to the environment characterization subsystem, which returns a list of matched features (if any are found), as well as two lists of regions $R^1 = \{r_0^1, r_1^1, \ldots\}$, $R^2 = \{r_0^2, r_1^2, \ldots\}$ that the matched features connect.

To decide what structure to build, the high-level planner considers the available augmentation modules in the current environment, the current robot configuration, and the distance from the structure build-point to the robot goal position. After selecting a structure, the high-level planner executes its assembly plan to construct it. Once the structure is built, the high-level planner considers regions r_i^1 and r_i^2 to be connected and traversable by the robot, allowing it to complete its overall task of climbing onto the drawer.

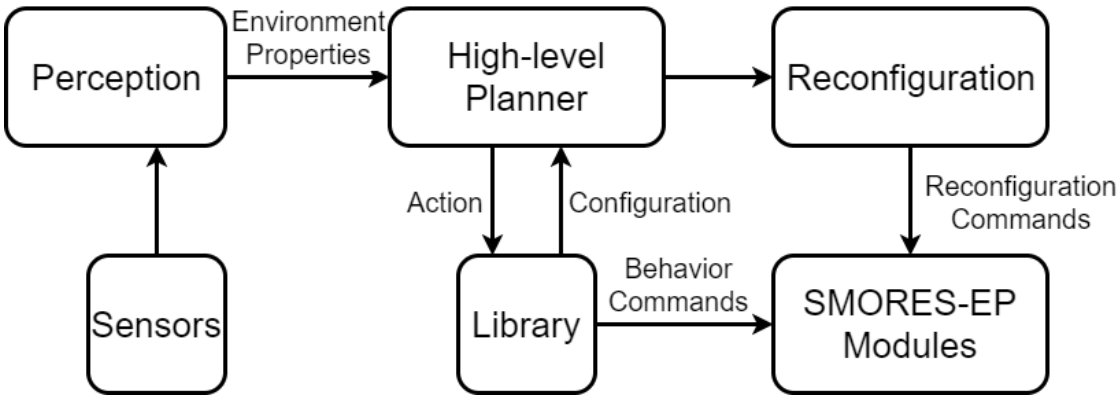


Figure 4.7: System Overview Flowchart

4.4 System Integration

We integrate our environment augmentation tools into the system introduced in [6], as shown in Figure 4.7. The high-level planner automatically converts user defined task specifications to controllers from a robot design library. It executes the controller by reacting to the sensed environment, running appropriate behaviors from the design library to control a set of hardware robot modules. Active perception components perform simultaneous localization and mapping (SLAM), and characterize the environment in terms of robot capabilities. Whenever required, the reconfiguration subsystem controls the robot to change configurations.

The system used the Robot Operating System (ROS)[2] for a software framework, networking, and navigation. SLAM was performed using RTAB-MAP[31], and color detection was done using CMVision[3].

SMORES-EP modules have no sensors that allow them to gather information about their environment. To enable autonomous operation, we use the *sensor module* shown in Figure 4.8. The sensor module has a 90mm × 70mm × 70mm body with thin steel plates on its front and back that allow SMORES-EP modules

[2]http://www.ros.org
[3]CMVision: http://www.cs.cmu.edu/∼jbruce/cmvision/

Figure 4.8: Sensor Module with labelled components. UP board and battery are inside the body.

to connect to it. Computation is provided by an UP computing board with an Intel Atom 1.92 GHz processor, 4 GB memory, and a 64 GB hard drive. A USB WiFi adapter provides network connectivity. A front-facing Orbecc Astra Mini RGB-D camera enables the robot to map and explore its environment and recognize objects of interest. A thin stem extends 40cm above the body, supporting a downward-facing webcam. This camera provides a view of a 1m $\times$ 0.75m area in front of the sensor module, and is used to track AprilTag [40] fiducials on modules and augmentation modules for reconfiguration and structure building. A 7.4V, 2200mAh LiPo battery provides about one hour of running time.

4.5 Experiment Results

Our system can generalize to arbitrary environment augmentations and high-level tasks. We validate our system in two hardware experiments that require the same system to perform tasks requiring very different environment augmentations for successful completion. In both experiments, the robot autonomously perceives and characterizes each environment, and synthesizes reactive controllers to accomplish

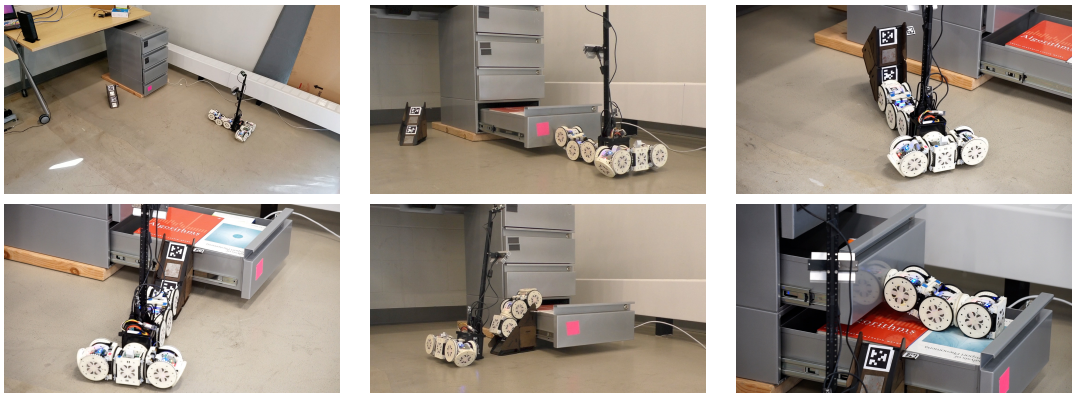


Figure 4.9: Snapshots throughout Experiment I. From left to right, top to bottom: i) Experiment start ii) Opening first drawer iii) Picking up ramp iv) Placing ramp next to open drawer. v) Reconfiguring and climbing ramp vi) Opening second drawer

the task based on the environment. Videos of the full experiments accompany this paper, and are also available online at https://youtu.be/NKj-xulsxco.

4.5.1 Experiment I

For the first experiment, the robot is tasked with inspecting two metal desk drawers in an office. It is asked to locate the set of drawers (identified with a pink label), open the two lowest drawers, and inspect their contents with its camera. The robot can open metal drawers by magnetically attaching to them and pulling backwards, but it is unable to reach the second drawer from the ground. Therefore, it can only open the second drawer if it can first open the bottom drawer and then climb on top of the things inside it.

Figure 4.9 shows snapshots throughout the robot's autonomous performance of Experiment I. After recognizing and opening the first drawer, the robot characterizes the environment with the opened drawer and identifies the side of the drawer as a "ledge" feature. The high-level planner recognizes that the ledge is too high

for the current configuration to climb, and furthermore that there is no other configuration in the library to which the robot can transform that could climb the ledge, leaving environment augmentation as the only strategy that can complete the task. Observing a ramp structure in the environment, the high-level planner commands the robot to acquire the ramp, place it at the "ledge" feature detected by the characterization algorithm, climb the drawer, and complete the mission.

In a second version of the same experiment, the first drawer is empty. When the robot characterizes the environment containing the drawer, it identifies no "ledge" features, since the drawer no longer matches the requirements of the feature. As a result, it recognizes that environment augmentation is not possible, and the mission cannot be completed.

4.5.2 Experiment II

The environment for Experiment II consists of two tables separated by a 16 cm gap. The robot begins the experiment on the left table with two wedges and one block. To complete its mission, the robot must cross the gap to reach a pink destination zone on the right table.

Figure 4.10 shows snapshots throughout Experiment II. This time, characterization of the environment identifies that the pink goal zone is in a separate region from the robot, and also identifies several "gap" features separating the two regions. Recognizing that the gap is too wide for any configuration in the design library to cross unassisted, the high-level planner concludes it must build a bridge across the gap to complete its mission. It begins searching for materials, and quickly identifies the three available augmentation modules, which it autonomously assembles

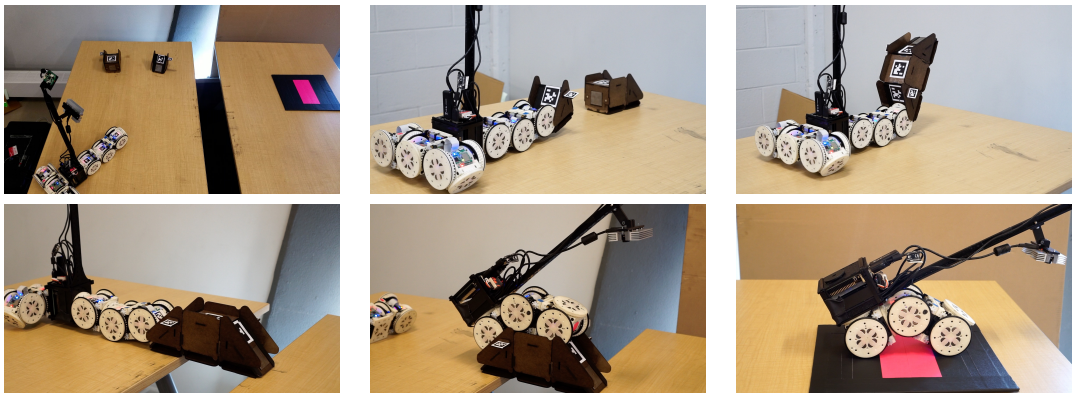


Figure 4.10: Snapshots throughout Experiment II. From left to right, top to bottom: i) Experiment start ii) Assembling bridge iii) Transporting bridge iv) Placing bridge over gap. v) Reconfigure and cross bridge. vi) Arrive at the target zone.

into a bridge. It then places the bridge across the gap and crosses to complete its mission.

4.6 Discussion

Block and wedge modules demonstrably expand the physical capabilities of SMORES-EP, allowing the system to climb to a high ledge and cross a wide gap to complete tasks that would have been very difficult with the SMORES-EP modules alone. Perception tools accurately characterize augmentable features in the environment. High-level reasoning tools identify when environment augmentation is necessary to complete a high-level task, and reactively sequence locomotion, manipulation, construction, and reconfiguration actions to accomplish the mission. The presented work represents the first time that modular robots have successfully augmented their environment by deploying passive structures to perform high-level tasks.

As with our previous work with autonomous modular robots [6], robustness

Outcome	Exp 1	Exp 2
Success	2 (25.0%)	3 (37.5%)
Perception-Related Failure	2 (25.0%)	2 (25.0%)
Navigation Failure	1 (12.5%)	0 (0.0%)
Hardware Failure	3 (37.5%)	2 (25.0%)
Setup Error	0 (0.0%)	1 (12.5%)

Table 4.1: Outcomes for Experiments 1 and 2

proved challenging. Out of 8 test runs of Experiment 1, the robot successfully completed the entire task once. Table 4.1 shows the outcomes of 8 runs of each experiment. The largest source of error was due to hardware failures such as slight encoder mis-calibration or wireless communication failure. Creating more robust hardware for modular robots is challenging due to the constrained size of each module, and the higher probability of failure from higher numbers of components in the system.

Perception-related errors were another frequent cause of failure. These were due in part to mis-detections by the characterization algorithm, or because the accuracy in finding location and orientation of features was not high enough for the margin of error of the robot when placing structures. Finally, navigation failures occurred throughout development and experiments due to cumulative SLAM errors as the robot navigates the environment. We found that it was important to minimize in-place rotation of the robot, and to avoid areas without many features for visual odometry to use.

4.6.1 Future

In the interest of establishing the deployment of structures as an effective means to address high-level tasks, this work does not focus on the speed or scale of construction, demonstrating the use of only small structures (with three elements).

Future work might attempt to accelerate construction, build larger structures, and attempt larger-scale tasks with SMORES-EP. For the purposes of this work, structure assembly plans (A_n) were create manually, but this process could be automated by employing established assembly planning algorithms [49, 59]. Assembly might be significantly accelerated by using multiple builders in parallel, as some other collective robot construction systems have done [43]. To truly scale to large tasks, a large number of block and wedge modules must be available in the environment, or better, autonomously transported into the environment. Developing mechanisms for transporting building material to a task location remains an open challenge for future work.

While our system implementation is tightly coupled to the SMORES-EP hardware, the concepts, system architecture, and theoretical frameworks could be applied widely. In particular, most elements of the framework could be directly applied to the Termes [43] or foam-ramp building robots [37] that have similar construction and locomotion capabilities to SMORES-EP, provided that appropriate sensing and perception capabilities were established.

4.6.2 Conclusion

To conclude, this paper presents tools that allow a modular robot to autonomously deploy passive structures as a means to complete high-level tasks involving locomotion, manipulation, and reconfiguration. This work expands the physical capabilities of the SMORES-EP modular robot, and extends our existing frameworks for addressing high-level tasks with modular robots by allowing both the robot morphology and the environment to be altered if doing so allows the task to be completed. We validate our system in two hardware experiments that demon-

strate how the hardware, perception tools, and high-level planner work together to complete high-level tasks through environment augmentation.

CHAPTER 5

A MULTI-MODAL OPTIMAL PATH PLANNER FOR GENERIC MODULAR ROBOTS ON DIFFICULT TERRAIN

5.1 INTRODUCTION

Modular self-reconfigurable robot (MSRR) systems consist of multiple robotic elements, called modules, that connect to each other to form larger robotic morphologies. They also have the ability to self-reconfigure to a different morphology with different shapes and capabilities. This capability also enables MSRRs to have multiple types of locomotion, corresponding to different morphologies. An MSRR can traverse challenging terrain by reconfiguring to adapt its shape and gait (e.g. walking, rolling, etc.) to the terrain. For example, a ledge or narrow corridor may not be traversable by a simple wheeled robot. However, an MSRR may be able to reconfigure to a snake-like morphology that can climb the ledge or squeeze through the corridor. The MSRR can then reconfigure back to a car-like morphology to traverse flat terrain optimally. Figure 5.1 shows three example morphologies of simulated SMORES-EP modules. [7]

In order to leverage this adaptive potential in an autonomous MSRR system, a planning framework is required that can (a) characterize the environment in terms of the navigation abilities of the robot, and (b) plan paths and morphologies in the environment, including reconfiguration between morphologies when necessary. These paths must take into account the locomotion abilities of each robot morphology, and the ability to reconfigure between morphologies.

To satisfy these requirements, we present the Multi-Morphology Module Plan-

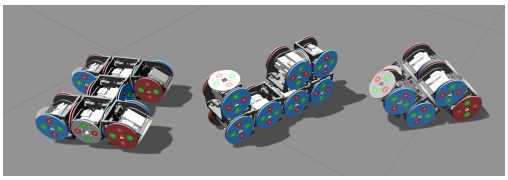


Figure 5.1: Three example MSRR morphologies. A *car* (left), *snake* (center), and *dolphin* (right). Each morphology has a unique set of locomotion capabilities, and can reconfigure to one of the other morphologies.

ner (MMM Planner): a novel, end-to-end planning framework for MSRRs. The MMM Planner consists of two parts: terrain characterization and path planning with reconfiguration. Terrain characterization uses a convolutional neural network (CNN) to create a 2D semantic grid representation of the terrain in terms of discrete *terrain features*: small regions of the terrain known to be traversable by at least one robot morphology using pre-defined action primitives. The 2D semantic grid is called a feature map. The planner uses the resulting feature map to plan optimal paths with reconfiguration to desired goals in the environment. By using generic definitions of terrain features and action primitives, our framework is generalizeable to any MSRR system. A human expert can setup our framework for an MSRR system by labeling examples of terrain features to train the terrain characterization network, and by defining action primitives for each feature to specify how each robot morphology can move in the environment. We use a 4D lattice planner with a state consisting of the robot's (x, y) position, heading, and configuration to plan optimal paths in the environment.

The remainder of this paper is organized as follows. Section II presents related work from the field. Section III, IV, and V present the terrain characterization and path planning algorithms. Section VI presents the implementation of MMM Planner on a simulated MSRR system and Section VII describes the experimental setup and results. Section VIII concludes the work.

5.2 RELATED WORK

Several MSRR systems have been developed with the ability to traverse difficult terrain by reconfiguring their morphologies and using multiple types of gaits [7] [66] [63] [16] [51]. As the mechanical and low-level control abilities of MSRR systems improve, it is important to develop intelligent perception, planning, and control tools to leverage these abilities for performing autonomous tasks. One core component for making an autonomous MSRR system is a planning framework that enables the robot autonomously plan optimal paths with reconfiguration to reach goals in the environment based on sensing and perception of the terrain. To leverage the unique ability of MSRR systems to adapt to difficult terrain via reconfiguration, such a planning framework should be able to reason about the available robot morphologies and gaits when searching for optimal paths.

There is existing work in the field of path planning for single-morphology robots that can handle rough terrain, such as legged robots. Some formulate a "traversability" estimate of the terrain at each point in a grid of the environment, using measures such as surface slope and roughness [4] [60]. These planners are designed for robots with a fixed morphology, and use simple formulations of traversability that do not generalize to robots that can traverse more complex types of terrain.

Several works have created machine learning-based classifiers to characterize terrain in terms of its traversability. For example, [48] uses a Fully Convolutional Network [50] to perform pixel-wise semantic segmentation of the terrain into safe, risky, or dangerous regions. Bartoszyk *et al.* uses an SVM to classify terrain from monocular images and assigns a traversability cost to each location based on the terrrain class [2]. As with [4] and [60], this work is applicable for a robot with a

single morphology, such as a legged robot.

Daudelin *et al.* present a fully autonomous MSRR system for performing high-level tasks in unknown environments [5]. It includes environment characterization with reactive controller synthesis to determine if the robot needs to reconfigure during task performance based on perception of the environment. However, it does not perform optimal planning that includes reconfiguration. For navigation, the system uses an off-the-shelf navigation package that plans only in flat space using a path planner for wheeled robots; environment characterization is considered only at points of interest to inform reconfiguration and other reactive behaviors. The work in this paper could serve as an improvement to the system in [5], by replacing the flat-space navigation package with one that optimizes for reconfiguration to handle more challenging terrain.

A few groups have created planners for modular robots. Tonglin *et al.* present a local motion planner that takes into account the footprints of multiple robot morphologies [32]. Only flat terrain is considered, however, and the planner cannot find globally optimal paths. Klidbary *et al.* present a global path planner for modular robots using a grid planner with "super nodes": super nodes at each 2D location in the planning graph contain nodes for each morphology with edges between them to enable reconfiguration [17]. However, the terrain is assumed to be characterized *a priori*, and highly-restrictive assumptions are made about the simulated environment that are not applicable in real-world scenarios. For example, the planner assumes all robot morphologies can move in eight directions for all characterized types of terrain. Thus, the environments in their simulated results are defined in regular grids with each type of terrain oriented at one of the 8 directions of robot motion. Our framework removes these simplifying assumptions,

allowing for arbitrary definitions of terrain features and action primitives for each robot morphology, making it applicable for the real world and generalizeable to any MSRR system.

5.3 Path Planning with Reconfiguration using Lattice Graph Search

The proposed MMM Planner plans trajectories over a 4-dimensional robot state $s = (x, y, \theta, \mu)$, describing the robot position (x, y), orientation θ, and morphology μ. To find optimal paths through the environment with reconfiguration, the MMM planner uses a state lattice to discretize the state space. The state lattice is stored in a directed, cyclic graph, where each vertex represents a set of discrete state values $s_i = (x_i, y_i, \theta_i, \mu_i)$ in the environment. Vertices are connected by edges representing valid robot actions, including both motion and reconfiguration. Edges are assigned nonnegative costs representing the time required to perform the corresponding action. Once the graph is created and populated with edges for the robot's environment, a Dijkstra graph search algorithm is used to plan optimal paths with reconfiguration from the initial robot state to target states in the environment. Lattice planners have been successfully used with 4D states with real-time performance in other applications such as planning for autonomous cars. [12]

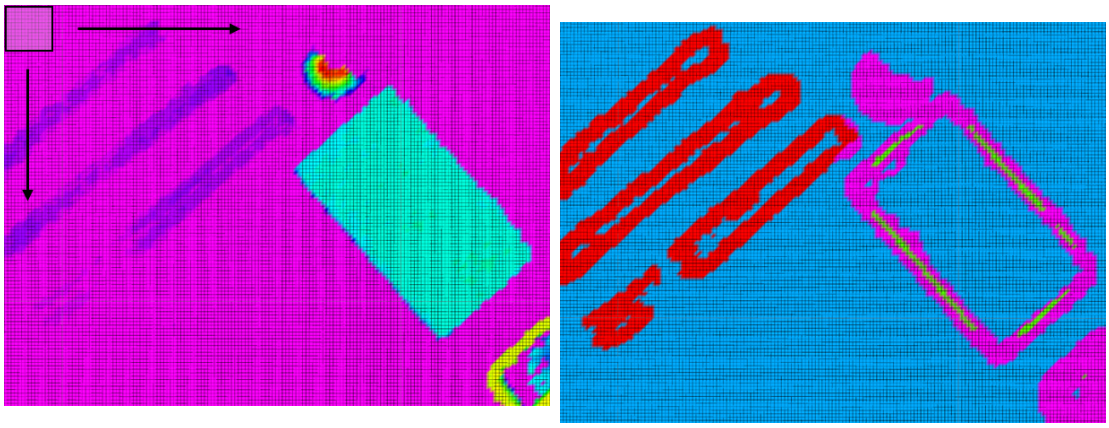
5.4 Terrain Characterization for MSRRs

Given a partial or complete global height map of the environment obtained from sensor measurements such as LIDAR or stereo vision, the first step of the MMM Planner is to characterize the terrain into a 2D semantic grid of *terrain features*. A terrain feature is a characterization of a 2D region of the environment. Each terrain feature corresponds to a set of valid robot actions that can be executed in regions of the environment with that characterization.

Terrain features provide a method of linking discretized regions of the environment to robot morphologies and actions that enable traversal of those regions. For example, a feature corresponding to steep ledges may be traversable by a *snake* morphology executing a *climbing* action. A region that cannot be traversed by any morphologies corresponds to the empty set. Terrain features may have an associated orientation (for example, the orientation of the normal to the face of a ledge), or can be orientation-agnostic (such as flat or bumpy terrain).

MMM Planner uses a convolutional neural network (CNN) to detect terrain features. This provides a rotation-invariant, general method of detecting terrain features that can be defined for any MSRR system via training by a human expert.

To characterize the terrain, the height map of the observed portion of the environment is converted to a greyscale image, with pixel intensity corresponding to terrain height. Then a sliding window CNN is used to classify each grid cell as a terrain feature or an obstacle, as shown in Figure 5.2a. The inputs to the network are 18 x 18 windows of the image of the global height map. As the sliding window moves over the image, the network classifies each window with the label of one of the terrain features or as an obstacle. This results in a semantic, 2D "feature map"



(a) Height map of terrain.

(b) Characterized feature map.

Figure 5.2: Characterized feature map of terrain (right) from a height map (left). Blue, red, green, and pink cells indicate *flat*, *wavy*, *ledge*, and *obstacle* features, respectively.

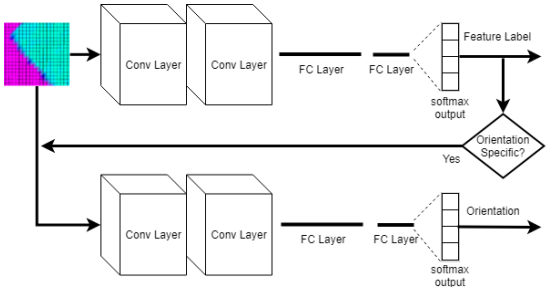


Figure 5.3: The network architecture for terrain characterization.

of the environment $\mathcal{F}(x, y) = n \in \mathbb{N}$, where n is the label of the terrain feature at grid cell (x, y). Figure 5.2b shows an example feature map. If the detected feature is orientation-specific, a secondary CNN is used to predict the orientation of the feature. Figure 5.3 shows the network architecture. Each CNN consists of two convolutional layers, two fully connected layers, and a softmax layer to obtain classification weights. The label with maximum weight is selected as the output.

To train the network, MMM Planner includes a tool that allows a human to quickly hand-label training examples from sensor data using the rviz visualization tool in the Robotics Operating System (ROS).[1] After loading a previously recorded

[1] http://www.ros.org

height map of training terrain, the user clicks on locations on the map to extract training samples. The user also specifies the orientation of the feature, and enters the correct label of the sample. The tool then generates many 18 x 18 depth images centered at this point, rotating the image at various orientations and offset heights to generate additional training data to make the classifier rotational and height invariance. However, the orientation of each training sample is stored and used to train the secondary classifier that determines the orientation of each detected feature.

Although the CNN could be used to detect all terrain features, classification of flat regions of terrain is detected separately using a fast flatness calculator. This is done to improve computation speed since most MSRR systems are likely to use flat terrain as a terrain feature and it is easy to define algorithmically.

5.5 Graph Creation

The MMM Planner uses a 4D lattice graph to find optimal paths with reconfiguration from start to goal states. Algorithm 2 defines the method for creating and connecting the graph for planning.

Algorithm 2 first creates vertices $\mathcal{V}$ in the graph $\mathcal{G}$ using a function CreateVertices(X, Y, Θ, M), which creates vertices to represent every discretized state in the state space. The terms X, Y, Θ, and M denote the cardinalities of the state space dimensions x, y, θ, and μ. Next, edges are created based on the dynamic capabilities of the MSRR under the environmental constraints. MSRR capabilities are represented using a set of *action primitives* $\mathcal{A} = \{a_1, a_2, ...\}$, where an action primitive $a : \mathbb{R}^2 \rightarrow \mathbb{R}^4 \times \mathbb{R}^4$ is a function defined as follows:

Algorithm 2 Graph Creation

Require: X, Y, Θ, M
Require: Action Set $\mathcal{A}$
$\mathcal{V}, \mathcal{E} = \emptyset$
$\mathcal{V} \leftarrow$ CreateVertices(X, Y, Θ, M)
for all $v \in \mathcal{V}$ **do**
  $s_0 = (x_0, y_0, \theta_0, \mu_0) \leftarrow v$
  for all $a \in \mathcal{A}$ **do**
    $(s_1, s_2) = a(x_0, y_0)$
    if $s_1, s_2 \in \mathcal{V}$ **and** IsEdgeValid(x_0, y_0, s_1, s_2, a) **then**
      $\mathcal{E} \leftarrow (s_1, s_2)$
    end if
  end for
end for
Graph $\mathcal{G} = (\mathcal{V}, \mathcal{E})$

$$a(x_0, y_0) = (s_1, s_2) \tag{5.1}$$

where (x_0, y_0) denotes the location of a terrain feature, and s_1 and s_2 represent start and end states of a discrete robot action (such as a locomotion trajectory or reconfiguration) that can traverse that terrain feature. These actions are defined offline from the MSRR system's dynamic capabilities. At each 2D location in the graph, each action primitive has the potential to create a single edge between two nearby vertices corresponding to states s_1 and s_2.

Before creating an edge between vertices corresponding to the states linked by an action primitive, the algorithm checks if the action primitive is valid in that region of the environment; i.e. if the robot can execute the action over the type of terrain in that region. This is done using the function IsEdgeValid(x_0, y_0, s_1, s_2, a), defined in Algorithm 3. This function uses a pre-defined set of constraints for each action primitive:

$$\mathcal{C}(a) = (f, \vec{\mathbf{o}}_1, \vec{\mathbf{o}}_2) \tag{5.2}$$

where f is the terrain feature that must exist at the location of the action, (x_0, y_0). $\vec{\mathbf{o}}_1 = \{f_1^1, f_1^2, ..., f_1^n\}$ is a vector of binary variables $f_1^i \in \{0, 1\}$ which indicates which of the n terrain features are not allowed within the robot's footprint at state $s1$ for the action to be valid: a value $f_1^i = 1$ indicates that terrain feature i cannot be contained in the robot's footprint at state $s1$. With a similar definition for $\vec{\mathbf{o}}_2$), these two constraints impose conditions on the terrain at the start and end states of each action primitive. Validity can be checked by comparing $\vec{\mathbf{o}}_1$ and $\vec{\mathbf{o}}_2$ with binary vectors indicating which terrain features are present at the locations of s_1 and s_2. To obtain these, MMM Planner creates an *enlarged feature map* as described in the next section.

Algorithm 3 IsEdgeValid(x_0, y_0, s_1, s_2, a)

Require: Feature Map $\mathcal{F}$
Require: Enlarged Feature Map $\mathcal{F}_E$
Require: Constraint Set $\mathcal{C}$
1: $f, \vec{\mathbf{o}}_1, \vec{\mathbf{o}}_2 \leftarrow \mathcal{C}(a)$
2: **if** $\mathcal{F}(x_0, y_0) == f$ **and** $\mathcal{F}_E(s_1) \leq \vec{\mathbf{o}}_1$ **and** $\mathcal{F}_E(s_2) \leq \vec{\mathbf{o}}_2$ **then**
3: **return true**
4: **else**
5: **return false**
6: **end if**

5.5.1 Feature Enlargement for Obstacle Avoidance

Graph planners for wheeled robots often assume a circular robot radius and consider each grid cell as one of only two terrain features: "obstacle" or "free" (such as [**?**]). [**?**] enlarges obstacles by this radius to create an enlarged obstacle map of the environment to avoid collision with obstacles during planned paths. MMM

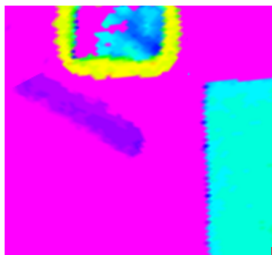
Planner extends this principle for our more general case of an arbitrary number of terrain features, and an arbitrary number of robot morphologies and orientations (corresponding to different footprints).

For n terrain features, we define an *enlarged feature map* $\mathcal{F}_E(x, y, \mu, \theta) = \{f_1, f_2, ..., f_n\}$, where $f_i \in \{0, 1\}$ is a binary variable denoting the presence of terrain feature i within the footprint of robot morphology μ at grid cell location (x, y) with an orientation of θ.

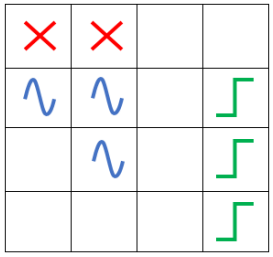
To create $\mathcal{F}_E(x, y, \mu, \theta)$, an enlarged feature submap $\mathcal{F}_E^{\mu,\theta}(x, y)$ is created for each μ and θ by enlarging each terrain feature in the feature map $\mathcal{F}$ by the footprint of μ at orientation θ. Rectangular footprints for each morphology are predefined by a human when setting up MMM Planner for an MSRR system. Figure 5.4 describes an illustrative example of this process. Figure 5.4a shows the raw heightmap, which results in the feature map in Figure 5.4b. Figure 5.4c and Figure 5.4d show the enlarged feature submaps $\mathcal{F}_E^{\mu,\theta}$ for two morphologies with different footprints. The small icons on the left side of each grid cell indicates the presence of each feature in the footprint of the oriented morphology at that location.

Creation of $\mathcal{F}_E$ needs to be performed only once for a given map of the environment, and does not need to be repeated when planning new paths. In addition, $\mathcal{F}_E$ can be recursively updated when the environment map is updated.

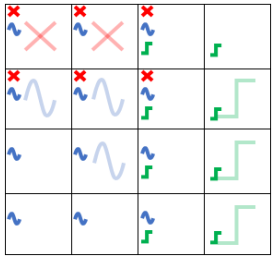
An additional enlarged feature submap $\mathcal{F}_E^{\mu,\theta}$ is created for reconfiguration actions, using a circular footprint with a radius pre-defined by the space requirements of the MSRR's reconfiguration controller.



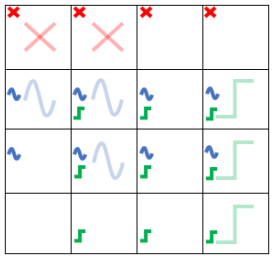
(a) Height map for toy example.



(b) Resulting feature map. The large icons in the center of grid cells indicate the presence of "wavy", "ledge", or "obstacle" features in the feature map, while the absence of an icon indicates a "flat" feature.



(c) Enlarged feature submap for a morphology with a 3 x 3 footprint at an orientation of 0°.



(d) Enlarged feature submap for a morphology with a 5 x 1 footprint at an orientation of 0°.

Figure 5.4: Enlarged feature map example.

5.5.2 Computational Complexity

Once the planning graph has been constructed, Dijkstra's search algorithm is used to find shortest paths to goal states. The graph creation operation has complexity $\mathcal{O}(X \cdot Y \cdot \Theta \cdot M)$. Dijkstra's search complexity has a strict upper bound of $\mathcal{O}(V^2)$, where $V = X \cdot Y \cdot \Theta \cdot M$ is the number of nodes in the graph. However, practical bounds given sparsity of edges in path planning applications such as ours are

$\mathcal{O}(V \cdot \log(V))$. Search algorithms such as A* and D* introduce heuristics to improve computation speed further in practice. However, this work uses Dijkstra's search algorithm to demonstrate the real-time ability of our approach even with basic optimization methods.

5.6 Implementation on SMORES-EP System

We implemented the MMM Planner on a simulated version of the SMORES-EP system [7], using the morphologies shown in Figure 5.1. The *car* morphology has differential-drive dynamics and therefore is well-suited for navigation across flat terrain, but cannot traverse any other terrain type. By using a "flapping" gait, the *dolphin* morphology can traverse *wavy* terrain in addition to $flat$, but can only move in a straight line forwards or backwards. Similarly, the *snake* morphology can only move in a straight line, but has the ability to climb *ledge* features using a "climbing" gait in addition to traversing on flat ground. The *snake* morphology has the additional advantage of having a thin footprint, enabling it to fit through more restricted spaces than the *car* or *dolphin* morphologies.

These locomotion and reconfiguration capabilities are defined in MMM Planner by creating an action set $\mathcal{A} = \{a_1, a_2, ...\}$ to describe each action ability, along with the constraint set $\mathcal{C}(a_i)$ to describe the required terrain conditions for each action to be valid. The terrain characterization system was trained on data recorded from real world environments to detect the following set of terrain features: $\{obstacle, flat, ledge, wavy\}$. We populated the action set with action primitives encoding locomotion capabilities for each morphology, as well as reconfiguration between morphologies. For example, one locomotion ability is a

"flapping" action that can be executed by the *dolphin* morphology to maneuver across *wavy* terrain features. One action primitive to encode this ability for a robot oriented at 0° is defined as follows:

$$a_{flap}\big((x_0, y_0)\big) = \big((x_0, y_0, 0°, \text{“}dolphin\text{”}), \\ (x_0 + 0.1, y_0, 0°, \text{“}dolphin\text{”})\big) \tag{5.3}$$

And the corresponding constraint:

$$\mathcal{C}(a_{flap}) = (\text{“}wavy\text{”}, [1, 0, 1, 0], [1, 0, 1, 0]) \tag{5.4}$$

The above action primitive encodes the fact that a robot with the *dolphin* morphology at 0° orientation can move from its current location to a location 0.1 meters in the positive x-direction. The constraint $\mathcal{C}(a_{flap})$ imposes the restrictions that an edge can only be created for this action primitive at a location corresponding to a *wavy* terrain feature, if the start and end robot states do not contain any *obstacle* or *ledge* features under the robot's footprint. Recall that $\vec{\mathbf{o}}_1 = \vec{\mathbf{o}}_2 = [1, 0, 1, 0]$ corresponds to the terrain feature list $\{obstacle, flat, ledge, wavy\}$, indicating that *obstacle* and *ledge* features cannot exist under the robot footprints at the two end states for an edge to be created. Figure 5.5 illustrates examples of this action primitive being checked for validity in creating edges in the graph (assuming a robot footprint of 3 x 3 grid cells). In these examples, an edge would be created between the start and end states of the action primitive (indicated by the arrow) for the left example, but not for the right example because an *obstacle* feature exists in the robot's footprint at state s_2, violating the constraint $\vec{\mathbf{o}}_2 = [1, 0, 1, 0]$.

Figure 5.6 shows another action primitive for an action primitive that allows

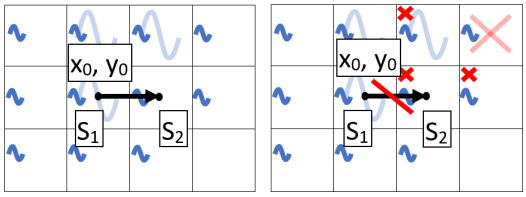


Figure 5.5: Examples of a valid (left) and invalid (right) action corresponding to a "flapping" action for the "dolphin" morphology. The black arrow represents the candidate edge between two lattice states s_1 and s_2 created by the action primitive.

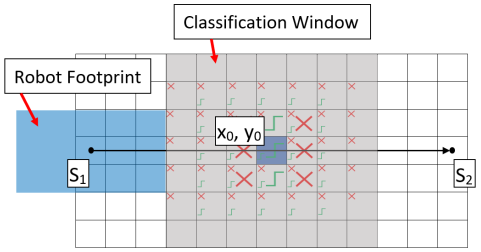


Figure 5.6: An example of a ledge-climbing action primitive connecting two states neighboring a "ledge" feature.

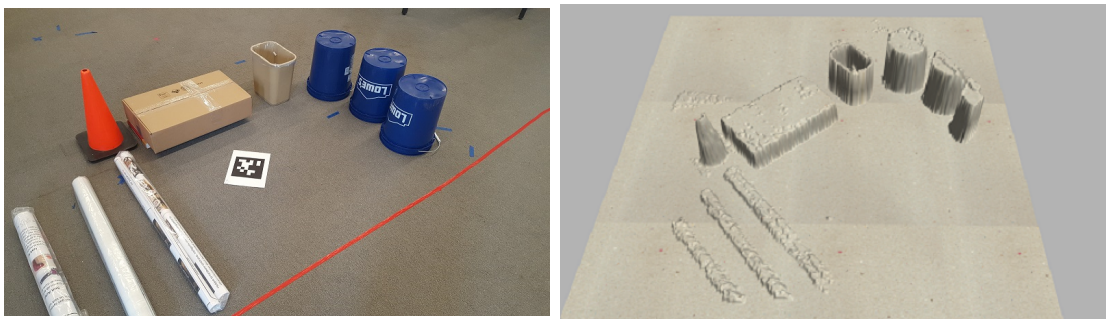
the "snake" morphology climb a ledge. Note that some grid cells bordering the detected *ledge* features were classified as *obstacle* features. However, since the enlarged feature map at the start and end states is free of all features other than $flat$, a valid edge is still created from the action primitive.

In all, the action set includes the following action primitives. For $flat$ features, actions link 8-connected neighbors for the *car* morphology (no orientation in the state). They also link forward and backward neighbors for the *dolphin* and *snake* morphologies at each orientation. All these actions require start and end states with only $flat$ features under the robot footprint. The *dolphin* morphology can traverse *wavy* terrain using the example forward-moving "flapping" gait. Thus, for *wavy* features, action primitives create edges to forward neighbors for the *dolphin* morphology at each orientation. The start and end states for these actions can contain either $flat$ or *wavy* features under the robot footprint. Finally, the *snake*

morphology has the ability to climb *ledge* features if the terrain on either side is *flat*. Thus, action primitives for *ledge* features at each *ledge* orientation link states on the low side of the ledge to a state on the high side of the ledge, requiring both states to have only *flat* terrain features under the robot footprint (as illustrated in Figure 5.6).

5.7 Hybrid Data in Simulation Results

To demonstrate the performance, real-time implementation, and robustness of our framework, two simulation-based experiments were tested using Gazebo[2] and the simulated SMORES-EP robot system. To create realistic experiments, raw data was collected from test environments in the laboratory. A Microsoft Kinect sensor using Apriltags [41] for localization was used to create a heightmap of the environment. This measured heightmap was used as the input to MMM Planner, and was imported into Gazebo as a simulated terrain to test the robot's navigation for Experiment I. The true and simulated versions of the environment are shown in Figure 5.7.

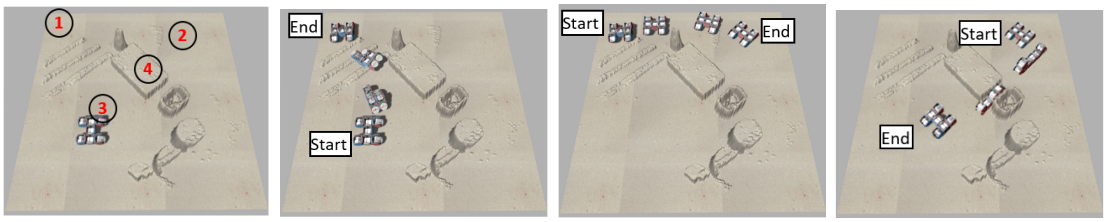


(a) Environment setup in lab.

(b) Measured height map of envronment simulated in Gazebo.

Figure 5.7: Environment setup for Experiment I.

[2]http://www.gazebosim.org



(a) Sequence of goal waypoints for navigation experiment.

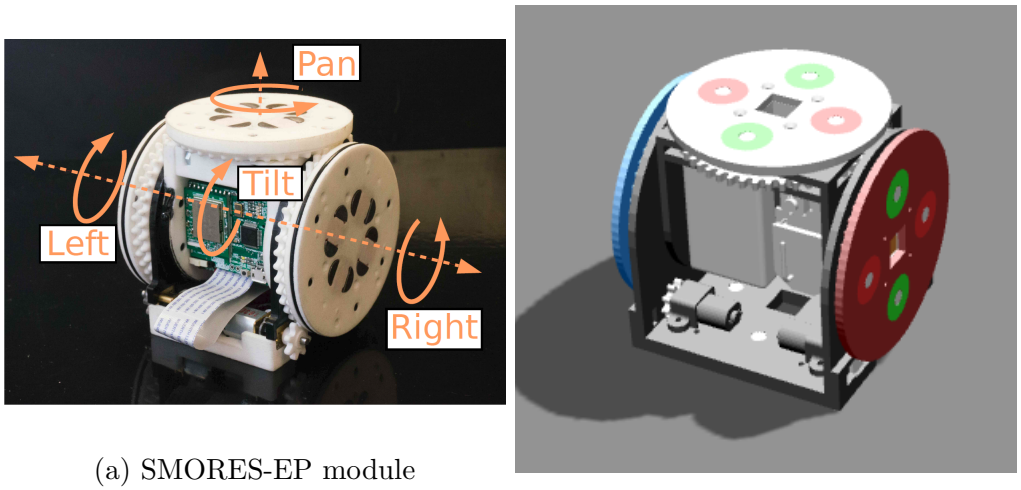
(b) Path to first waypoint.

(c) Path to second waypoint.

(d) Path to third waypoint.

Figure 5.8: Robot navigating to each goal waypoint during experiment I.

Figure 5.9 shows a SMORES-EP module and its simulation counterpart. The SMORES-EP module has 4 degrees of freedom, including two drive wheels that enable each module to locomote individually, a pan wheel, and a tilt degree of freedom that tilts the pan wheel inside the main module frame. Electro-permanent magnets enable modules to autonomously form or break connections with each other to reconfigure between robot morphologies. Since the reconfiguration process is not a focus of this work (but was included in [5]), reconfiguration is simulated by deleting the old morphology and spawning a new morphology at the same location.



(a) SMORES-EP module

(b) Simulated SMORES-EP module.

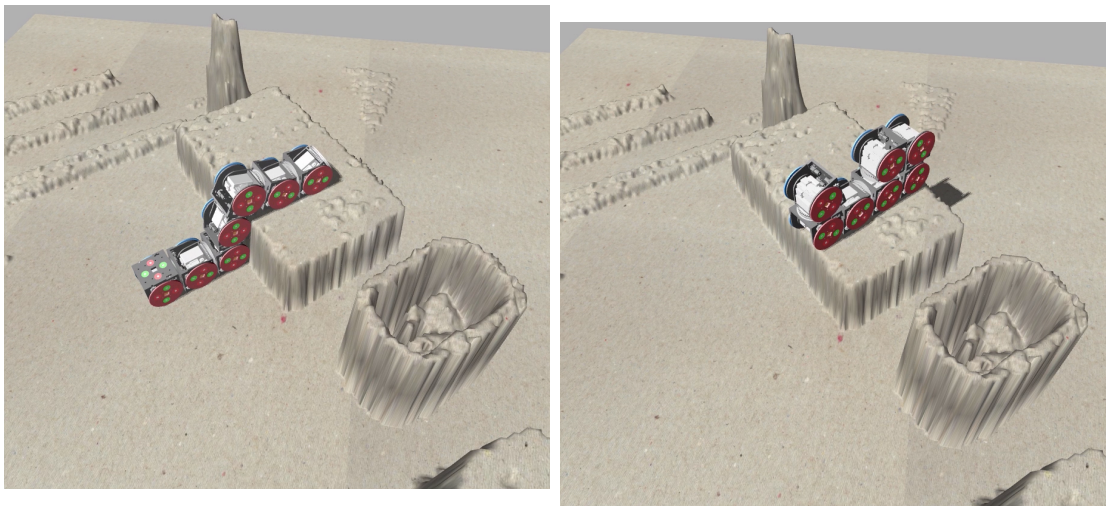
Figure 5.9: SMORES-EP Module and Sensor Module

5.7.1 Results

A video accompanying this submission shows the entire experiment processes. For Experiment I, the planner is directed to plan paths to a sequence of 4 navigation waypoints in the environment. Due to the obstructed nature of the environment, several of these waypoints require multiple morphologies and gaits to successfully reach the goal waypoint from the start state. Figure 5.8 shows the sequence of goal waypoints and snapshots of the robot travelling to the first three; Figure 5.10 shows snapshots of the robot climbing the ledge to reach the final goal waypoint. The robot successfully reaches all waypoints using the optimal path that includes switching between morphologies and using various gaits to traverse difficult terrain. For example, for the first goal waypoint, no path exists between the start and goal states without reconfiguration, due to the obstacles on all sides of the robot. However, the wavy terrain in the top left corner of the environment can be traversed by reconfiguring to the *dolphin* morphology and executing the "flapping" action to cross the terrain. Longer routes involving different reconfigurations can also reach the waypoint, but MMM Planner correctly chooses the former option since the total cost of locomotion and reconfiguration is lower.

Figure 5.11 shows a heatmap and corresopnding heightmap indicating the cost in total navigation time of the optimal path to each location in the environment from the specified starting location (starting in the *car* morphology). This heatmap reveals several observations about MMM Planner. First, the flat neighborhood near the starting location contains smoothly-changing navigation costs like a standard cost map for a wheeled robot. However, discontinuities occur in the vicinity of other features. This is due to the high cost of reconfiguration, a time-consuming operation. Notice also the very high costs associated with locations close to ob-

stacles such as the box in the middle of the environment. These costs result from the fact that only morphologies with thin footprints (i.e. the *snake* morphology) can reach these locations at highly oblique angles to the obstacle surfaces. Since the *snake* can only move straight without turning, reaching such a state requires the robot to first navigate to a point at which it can reconfigure to the *snake* morphology on the right line of incidence to the goal location, which in this environment often results in complicated and long paths, as demonstrated by the path in Figure 5.11b. However, this is still an improvement over such locations being entirely unreachable as for planners for standard wheeled robots.



(a) *Snake* morphology climbing a ledge to reach the last goal waypoint.

(b) Robot successfully reaches last goal waypoint.

Figure 5.10: Robot climbing a ledge.

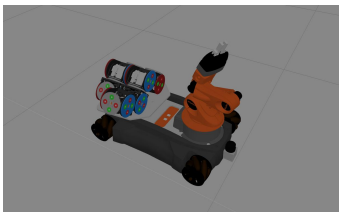


Figure 5.12: The *youBot* morphology.

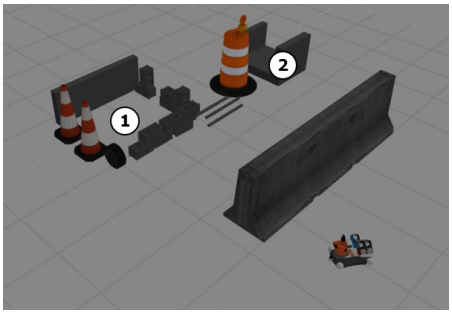
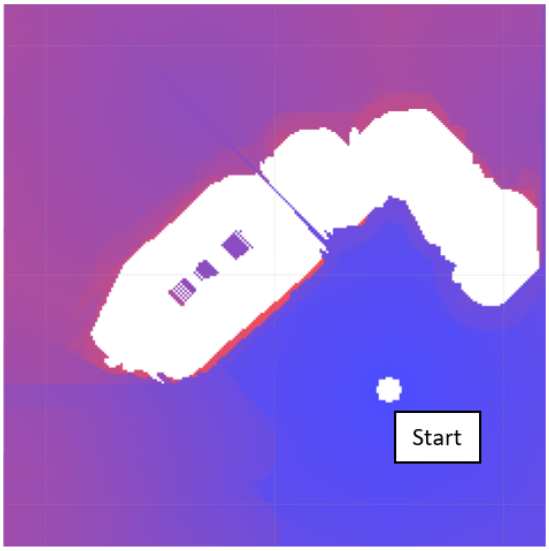
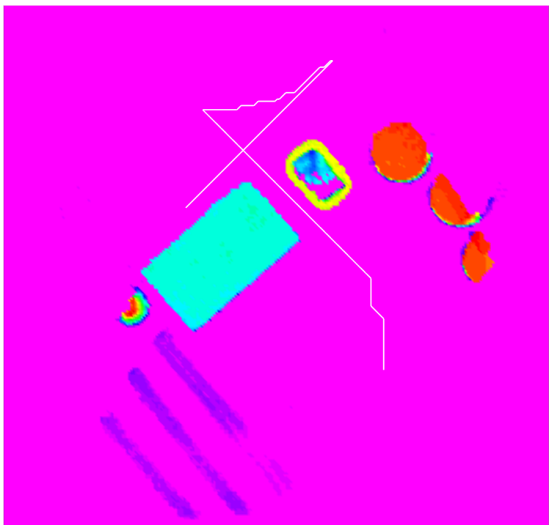


Figure 5.13: The environment and target waypoints used for Experiment II.



(a) Costmap of environment showing cost to navigate to each point in the environment.



(b) Height map of environment, along with a path from the start to an example goal waypoint.

Figure 5.11: A heatmap (left) of time costs to reach each location in the environment (right) from the given start location. Costs vary from blue (shorter time) to red (longer time). White areas indicate the location is unreachable.

We ran a second experiment to demonstrate how MMM Planner can be adapted

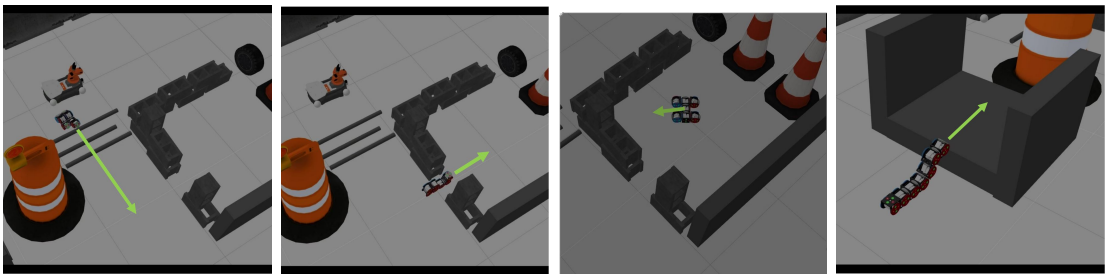


Figure 5.14: Chronological (left to right) snapshots of the robot's paths to each goal waypoint during Experiment II.

to different robot systems. For this experiment, we added a less conventional "morphology" consisting of a simulated KUKA youBot carrying the SMORES-EP modules in a mothership scenario (shown in Figure 5.12. The youBot is significantly faster than the SMORES-EP modules on flat terrain, but for more difficult terrain the youBot can deploy the SMORES-EP modules to navigate over more challenging terrain. To adapt MMM Planner to this expanded system, we simply added a *youBot* morphology with the footprint of the youBot, and action primitives/constraints to denote this morphology's ability to move over flat terrain similar to the *car* morphology, with appropriately lower costs for those actions. We also added an action primitive to describe a one-way reconfiguration from the *youBot* to the *dolphin* morphology next to the youBot's location.

Figure 5.13 shows the environment used for Experiment II. This environment is 9m x 9m (compared to the 2.6m x 2.6m environment from Experiment I), to test how MMM Planner scales computationally for larger environments. For this experiment, the robot navigated to the two goal waypoints shown in Figure 5.13. Figure 5.14 shows snapshots from the experiment. For the first waypoint, MMM Planner directs the youBot to drive a far as possible towards the goal waypoint since it is the fastest morphology. However, due to obstacles surrounding the waypoint, MMM Planner then directs the youBot to deploy the SMORES-EP modules in the *dolphin* morphology, which then traverses the wavy terrain. The robot then

reconfigures to the *snake* morphology in order to enter the walled area containing the waypoint. Finally, the robot reconfigures to the *car* configuration to precisely reach the waypoint. The second waypoint requires the robot to reconfigure to *snake* to exit the walled area, then to *car* to drive on flat ground to the front of the ledge containing the goal waypoint, and finally to *snake* to climb on top of the ledge and reach the goal waypoint. The robot successfully reaches both goal waypoints following optimal paths with reconfigurations found by MMM Planner.

For Experiment I, the environment was 2.57 m x 2.57 m at a grid resolution of 0.01 m. A total of 16 robot orientations and 3 morphologies were used. On a laptop equipped with an Intel Core i7 processor and 16 GB of RAM, terrain characterization took 0.37 seconds, and initial graph creation took 2.8 seconds. Both of these processes are performed once for a given global map. The actual path planning operation took 0.54 seconds, using the baseline Dijkstra algorithm and expanding all nodes in the graph. For Experiment II, we used a 9 m x 9 m environment at a grid resolution of 0.02 m, 16 orientations, and 4 morphologies. For this process, characterization took 1.4 seconds, graph creation took 19.8 seconds, and path planning took 4.5 seconds. The experiments demonstrated that our approach can be run in real-time.

5.8 Conclusion

A novel, end-to-end terrain characterization and path planning framework for generic modular robots has been presented. The planner is the first complete global path planning framework that leverages both autonomous reconfiguration and multiple-gait abilities of MSRR systems. The framework uses generalizeable

components such as feature classification and action primitives to make a general path planning tool for any MSRR system. Experiments in simulation - one using real data - demonstrate the effectiveness and real-time performance of our planner.

CHAPTER 6

FINAL REMARKS

The motivation of this work was to create active perception and planning frameworks to expand the autonomous capabilities of MSRR systems to leverage their adaptive capabilities to help perform high-level tasks and navigate challenging terrain. Here we summarize some key novelties and contributions of our work.

Chapter 2 presented an adaptable next-best-view object reconstruction algorithm for mobile robots. This novel algorithm is the first to adapt to objects of any size, making it useful for mobile robots exploring unknown objects in their environment. It is also designed to work well with modular robots, because it has the ability to leverage the potentially larger configuration space of MSRR systems to maximize reconstruction of unknown objects. In addition to scaling for objects of any size, the algorithm presented a novel information gain cost function that calculates the expected information gain from candidate viewpoints in a fully probabilistic fashion based on the current volumetric information collected on the object. It also presented novel methods for vastly improving computational speed of volumteric-based next best view algorithms, outperforming the speed of another state of the art algorithm by several orders of magnitude.

Chapters 3 and 4 presented fully autonomous MSRR systems for performing high-level tasks in an unknown environment. These systems represent a new frontier in autonomous MSRR technology as the first to explore an unknown environment and reactively reconfigure and synthesize new low-level controllers to adapt to perceived challenges in the environment in order to perform high-level tasks. My contributions to these projects include novel environment characterization algorithms to inform the system's high-level planner, and low-level reconfiguration

controllers to enable the robot to autonomously change its morphology.

Chapter 5 presented a novel, end-to-end system to perform terrain characterization and multi-modal path planning framework for generic MSRR systems. This framework is the first end-to-end optimal path planner for MSRR systems that characterizes terrain into modular capabilities and uses it to plan optimal paths that account for reconfiguration and multi-gait abilities of MSRRs. It also generalizes to any MSRR system.

BIBLIOGRAPHY

[1] J.E. Banta and M.A. Abidi. Autonomous placement of a range sensor for acquisition of optimal 3-d models. In *Industrial Electronics, Control, and Instrumentation, 1996., Proceedings of the 1996 IEEE IECON 22nd International Conference on*, volume 3, pages 1583–1588 vol.3, Aug 1996.

[2] S. Bartoszyk, P. Kasprzak, and D. Belter. Terrain-aware motion planning for a walking robot. In *2017 11th International Workshop on Robot Motion and Control (RoMoCo)*, pages 29–34, July 2017.

[3] C. Cai and S. Ferrari. A q-learning approach to developing an automated neural computer player for the board game of clue®. In *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*, pages 2346–2352, June 2008.

[4] A. Chilian and H. Hirschmüller. Stereo camera based navigation of mobile robots on rough terrain. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4571–4576, Oct 2009.

[5] Jonathan Daudelin*, Gangyuan Jing*, Tarik Tosun*, Mark Yim, Hadas Kress-Gazit, and Mark Campbell. An integrated system for perception-driven autonomy with modular robots. *In Preparation*, 2018.

[6] Jonathan Daudelin*, Gangyuan Jing*, Tarik Tosun*, Mark Yim, Hadas Kress-Gazit, and Mark Campbell. An integrated system for perception-driven autonomy with modular robots. *In Preparation*, 2018.

[7] Jay Davey, Ngai Kwok, and Mark Yim. Emulating self-reconfigurable robots-design of the SMORES system. In *IROS*, pages 4464–4469, 2012.

[8] F.-M. De Rainville, J.-P. Mercier, C. Gagne, P. Giguere, and D. Laurendeau. Multisensor placement in 3d environments via visibility estimation and derivative-free optimization. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 3327–3334, May 2015.

[9] Marco Dorigo, Dario Floreano, Luca Maria Gambardella, Francesco Mondada, Stefano Nolfi, Tarek Baaboura, Mauro Birattari, Michael Bonani, Manuele Brambilla, Arne Brutschy, Daniel Burnier, Alexandre Campo, Anders Lyhne Christensen, Antal Decugniere, Gianni Di Caro, Frederick Ducatelle, Eliseo Ferrante, Alexander Förster, Javier Martinez Gonzales, Jerome Guzzi, Valentin Longchamp, Stephane Magnenat, Nithin Mathews, Marco Montes

De Oca, Rehan O'Grady, Carlo Pinciroli, Giovanni Pini, Philippe Rétornaz, James Roberts, Valerio Sperati, Timothy Stirling, Alessandro Stranieri, Thomas Stützle, Vito Trianni, Elio Tuci, Ali Emre Turgut, and Florian Vaussard. Swarmanoid: A novel concept for the study of heterogeneous robotic swarms. *IEEE Robotics and Automation Magazine*, 20(4):60–71, 2013.

[10] Marco Dorigo, Elio Tuci, Roderich Groß, Vito Trianni, Thomas Halva Labella, Shervin Nouyan, Christos Ampatzis, Jean-Louis Deneubourg, Gianluca Baldassarre, Stefano Nolfi, Francesco Mondada, Dario Floreano, and Luca Maria Gambardella. The SWARM-BOTS Project. *LNCS*, 3342:31–44, 2005.

[11] E. Dunn, J. van den Berg, and J.-M. Frahm. Developing visual sensing strategies through next best view planning. In *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*, pages 4001–4008, Oct 2009.

[12] D. Ferguson, T. M. Howard, and M. Likhachev. Motion planning in urban environments: Part ii. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1070–1076, Sept 2008.

[13] Cameron Finucane, Gangyuan Jing, and Hadas Kress-Gazit. Ltlmop: Experimenting with language, temporal logic and robot control. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, October 18-22, 2010, Taipei, Taiwan*, pages 1988–1993, 2010.

[14] Robert Grabowski, Luis E. Navarro-Serment, Christiaan J J Paredis, and Pradeep K. Khosla. Heterogeneous teams of modular robots for mapping and exploration. *Autonomous Robots*, 8(3):293–308, 2000.

[15] Roderich Groß, Michael Bonani, Francesco Mondada, and Marco Dorigo. Autonomous self-assembly in swarm-bots. *IEEE transactions on robotics*, 22(6):1115–1130, 2006.

[16] Z. Guanghua, D. Zhicheng, and W. Wei. Realization of a modular reconfigurable robot for rough terrain. In *2006 International Conference on Mechatronics and Automation*, pages 289–294, June 2006.

[17] Sajad Haghzad Klidbary, Saeed Bagheri Shouraki, and Soroush Sheikhpour Kourabbaslou. Path planning of modular robots on various terrains using q-learning versus optimization algorithms. *Intelligent Service Robotics*, 10(2):121–136, Apr 2017.

[18] B.W. He and Y.F. Li. A next-best-view method with self-termination in active modeling of 3d objects. In *Intelligent Robots and Systems, 2006 IEEE/RSJ International Conference on*, pages 5345–5350, Oct 2006.

[19] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013.

[20] Armin Hornung, Kai M. Wurm, Maren Bennewitz, Cyrill Stachniss, and Wolfram Burgard. OctoMap: An efficient probabilistic 3D mapping framework based on octrees. *Autonomous Robots*, 2013.

[21] J. Irving Vasquez-Gomez, L.E. Sucar, and R. Murrieta-Cid. Hierarchical ray tracing for fast volumetric next-best-view planning. In *Computer and Robot Vision (CRV), 2013 International Conference on*, pages 181–187, May 2013.

[22] S. Isler, R. Sabzevari, J. Delmerico, and D. Scaramuzza. An information gain formulation for active volumetric 3d reconstruction. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3477–3484, May 2016.

[23] Gangyuan Jing, Tarik Tosun, Mark Yim, and Hadas Kress-Gazit. An End-To-End System for Accomplishing Tasks with Modular Robots. *Robotics: Science and Systems XII*, 2016.

[24] Gangyuan Jing, Tarik Tosun, Mark Yim, and Hadas Kress-Gazit. Accomplishing high-level tasks with modular robots. *Autonomous Robots*, 2017. Conditionally Accepted.

[25] M. Krainin, B. Curless, and D. Fox. Autonomous generation of complete 3d object models using next best view manipulation planning. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 5031–5037, May 2011.

[26] Hadas Kress-Gazit, Georgios E. Fainekos, and George J. Pappas. Temporal-logic-based reactive mission and motion planning. *IEEE Trans. Robotics*, 25(6):1370–1381, 2009.

[27] S. Kriegel, T. Bodenmliller, M. Suppa, and G. Hirzinger. A surface-based next-best-view approach for automated 3d model completion of unknown objects. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 4869–4874, May 2011.

[28] S. Kriegel, C. Rink, T. Bodenmüller, A. Narr, M. Suppa, and G. Hirzinger. Next-best-scan planning for autonomous 3d modeling. In *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2850–2856, Oct 2012.

[29] Simon Kriegel, Christian Rink, Tim Bodenmüller, and Michael Suppa. Efficient next-best-scan planning for autonomous 3D surface reconstruction of unknown objects. *Journal of Real-Time Image Processing*, 10(4):611–631, 2015.

[30] M. Labbe and F. Michaud. Online Global Loop Closure Detection for Large-Scale Multi-Session Graph-Based SLAM. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2661–2666, Sept 2014.

[31] M. Labbe and F. Michaud. Online Global Loop Closure Detection for Large-Scale Multi-Session Graph-Based SLAM. In *Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 2661–2666, Sept 2014.

[32] Tonglin Liu, Chengdong Wu, and B. Li. Shape-shifting robot path planning method based on reconfiguration performance. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4578–4583, Oct 2010.

[33] Stéphane Magnenat, Roland Philippsen, and Francesco Mondada. Autonomous construction using scarce resources in unknown environments. *Autonomous Robots*, 33(4):467–485, 2012.

[34] S. Maniatopoulos, P. Schillinger, V. Pong, D. C. Conner, and H. Kress-Gazit. Reactive high-level behavior synthesis for an atlas humanoid robot. In *2016 IEEE International Conference on Robotics and Automation (ICRA)*, May 2016.

[35] Francesco Mondada, Luca Maria Gambardella, Dario Floreano, Stefano Nolfi, Jean Louis Deneubourg, and Marco Dorigo. The cooperation of swarm-bots: Physical interactions in collective robotics. *IEEE Robotics and Automation Magazine*, 12(2):21–28, 2005.

[36] Satoshi Murata, Kiyoharu Kakomura, and Haruhisa Kurokawa. Docking experiments of a modular robot by visual feedback. *IEEE International Conference on Intelligent Robots and Systems*, pages 625–630, 2006.

[37] Nils Napp and Radhika Nagpal. Distributed amorphous ramp construction in unstructured environments. *Robotica*, 32(2):279–290, 2014.

[38] Nils Napp and Radhika Nagpal. Robotic construction of arbitrary shapes with amorphous materials. In *ICRA*, pages 438–444. IEEE, 2014.

[39] Rehan O'Grady, Roderich Groß, Anders Lyhne Christensen, and Marco Dorigo. Self-assembly strategies in a group of autonomous mobile robots. *Autonomous Robots*, 28(4):439–455, 2010.

[40] Edwin Olson. Apriltag: A robust and flexible visual fiducial system. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3400–3407. IEEE, 2011.

[41] Edwin Olson. Apriltag: A robust and flexible visual fiducial system. In *Robotics and Automation (ICRA), 2011 IEEE International Conference on*, pages 3400–3407. IEEE, 2011.

[42] James Paulos, Nick Eckenstein, Tarik Tosun, Jungwon Seo, Jay Davey, Jonathan Greco, Vijay Kumar, and Mark Yim. Automated Self-Assembly of Large Maritime Structures by a Team of Robotic Boats. *IEEE Transactions on Automation Science and Engineering*, pages 1–11, 2015.

[43] Kirstin Petersen, Radhika Nagpal, and Justin Werfel. Termes: An autonomous robotic system for three-dimensional collective construction. *Proc. Robotics: Science & Systems VII*, 2011.

[44] R. Pito. A solution to the next best view problem for automated surface acquisition. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 21(10):1016–1030, Oct 1999.

[45] Christian Potthast and Gaurav S. Sukhatme. A probabilistic framework for next best view estimation in a cluttered environment. *Journal of Visual Communication and Image Representation*, 25(1):148 – 164, 2014. Visual Understanding and Applications with RGB-D Cameras.

[46] John W Romanishin, Kyle Gilpin, Sebastian Claici, and Daniela Rus. 3d m-blocks: Self-reconfiguring robots capable of locomotion via pivoting in three dimensions. In *ICRA*, pages 1925–1932. IEEE, 2015.

[47] M. Rubenstein, K. Payne, P. Will, and Wei-Min Shen Wei-Min Shen. Docking among independent and autonomous CONRO self-reconfigurable robots.

IEEE International Conference on Robotics and Automation, 2004. Proceedings. ICRA '04. 2004, 3:2877–2882, 2004.

[48] F. Schilling, X. Chen, J. Folkesson, and P. Jensfelt. Geometric and visual terrain classification for autonomous mobile navigation. In *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 2678–2684, Sept 2017.

[49] Jungwon Seo, Mark Yim, and Vijay Kumar. Assembly planning for planar structures of a brick wall pattern with rectangular modular robots. *2013 IEEE International Conference on Automation Science and Engineering (CASE)*, pages 1016–1021, aug 2013.

[50] E. Shelhamer, J. Long, and T. Darrell. Fully convolutional networks for semantic segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 39(4):640–651, April 2017.

[51] G. Song, Xiaofeng Ye, Yanpeng Niu, and Tianhua Meng. A reconfigurable mobile node for wireless sensor networks in unfriendly environments. In *2010 The 2nd Conference on Environmental Science and Information Application Technology*, volume 1, pages 618–621, July 2010.

[52] Yuzuru Terada and Satoshi Murata. Automatic assembly system for a large-scale modular structure-hardware design of module and assembler robot. In *IROS*, volume 3, pages 2349–2355. IEEE, 2004.

[53] Tarik Tosun, Jay Davey, Chao Liu, and Mark Yim. Design and characterization of the ep-face connector. In *IROS*. IEEE, 2016.

[54] Tarik Tosun, Daniel Edgar, Chao Liu, Thulani Tsabedze, and Mark Yim. Paintpots: Low cost, accurate, highly customizable potentiometers for position sensing. In *ICRA*. IEEE, 2017.

[55] J I Vasquez-Gomez, L E Sucar, R Murrieta-Cid, and E Lopez-Damian. Volumetric Next Best View Planning for 3D Object Reconstruction with Positioning Error. *Int. J. of Advanced Robotic Systems*, 11:159, 2014.

[56] J. Irving Vasquez-Gomez, L. Enrique Sucar, and Rafael Murrieta-Cid. View/state planning for three-dimensional object reconstruction under uncertainty. *Autonomous Robots*, pages 1–21, 2015.

[57] Yiheng Wang, Alei Liang, and Haibing Guan. Frontier-based multi-robot map

exploration using particle swarm optimization. In *Swarm Intelligence (SIS), 2011 IEEE Symposium on*, pages 1–6, April 2011.

[58] S. Wenhardt, B. Deutsch, E. Angelopoulou, and H. Niemann. Active visual object reconstruction using d-, e-, and t-optimal next best views. In *Computer Vision and Pattern Recognition, 2007. CVPR '07. IEEE Conference on*, pages 1–7, June 2007.

[59] Justin Werfel, Donald Ingber, and Radhika Nagpal. Collective construction of environmentally-adaptive structures. *IEEE International Conference on Intelligent Robots and Systems*, pages 2345–2352, 2007.

[60] M. Wermelinger, P. Fankhauser, R. Diethelm, P. Krüsi, R. Siegwart, and M. Hutter. Navigation planning for legged robots in challenging terrain. In *2016 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1184–1189, Oct 2016.

[61] Jens Wettach and Karsten Berns. Dynamic frontier based exploration with a mobile indoor robot. In *Robotics (ISR), 2010 41st International Symposium on and 2010 6th German Conference on Robotics (ROBOTIK)*, pages 1–8, June 2010.

[62] L.M. Wong, C. Dumont, and M.A. Abidi. Next best view system in a 3d object modeling task. In *Computational Intelligence in Robotics and Automation, 1999. CIRA '99. Proceedings. 1999 IEEE International Symposium on*, pages 306–311, 1999.

[63] M Yim. *Locomotion with a unit-modular reconfigurable robot*. PhD thesis, Stanford University, 1994.

[64] M Yim, WM Shen, and Behnam Salemi. Modular self-reconfigurable robot systems: Challenges and Opportunities for the Future. *IEEE Robotics & Automation Magazine*, (March), 2007.

[65] Mark Yim, Kimon Roufas, David Duff, Ying Zhang, Craig Eldershaw, and Sam Homans. Modular reconfigurable robots in space applications. *Autonomous Robots*, 14(2):225–237, 2003.

[66] Mark Yim, Babak Shirmohammadi, Jimmy Sastra, Michael Park, Michael Dugan, and C J Taylor. Towards Robotic Self-reassembly After Explosion. In *IROS*, 2007.